UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Desarrollo de un “Datalogger” basado en ARM Cortex A9 y Linux mediante dispositivo Zynq-7000

Autor/es

ALBERTO OTAÑO JIMÉNEZ

Director/es

JAVIER ESTEBAN VICUÑA MARTÍNEZ

Facultad

Escuela Técnica Superior de Ingeniería Industrial

Titulación

Grado en Ingeniería Electrónica Industrial y Automática

Departamento

INGENIERÍA ELÉCTRICA

Curso académico

2018-19

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INDUSTRIAL

TRABAJO DE FIN DE GRADO

TITULACIÓN: Grado en Ingeniería Electrónica Industrial y Automática

CURSO: 2018/2019 **CONVOCATORIA: JULIO**

TÍTULO:

Desarrollo de un “Datalogger” basado en ARM Cortex A9 y Linux mediante dispositivo Zynq-7000

AUTOR: Alberto Otaño Jiménez

DIRECTOR/ES: Javier Esteban Vicuña Martínez

DEPARTAMENTO: Ingeniería Eléctrica

RESUMEN

Este Trabajo Fin de Grado está basado en el codiseño Hardware/Software de un PSoC (Programable System on Chip), formado por un sistema de procesamiento (PS) basado en un microprocesador ARM Cortex-A9 de doble núcleo, corriendo un sistema operativo Linux, y la lógica programable asociada (PL), todo ello implementado sobre un dispositivo de la familia Zynq-7000 de Xilinx. Ambas partes deben interactuar entre ellas, pero son bien diferenciadas. La parte lógica está programada en lenguaje VHDL y sigue la estructura de adquisición de datos, el envío de estos datos a un buffer FIFO y posteriormente al sistema de procesamiento. La parte PS está basada en el sistema microprocesador corriendo un operativo Xillinux, que permite el manejo sencillo de todas las interfaces de la tarjeta ZedBoard, como VGA, Ethernet, USB, etc.

Se ha desarrollado sobre la placa ZedBoard que incorpora el dispositivo XC7Z020-CLG484-1, de la familia Zynq-7000 de Xilinx. Además, incorpora una licencia del entorno de desarrollo Vivado, sobre el cual se trabajará la parte hardware. Sin embargo, la parte software se ha desarrollado en la distribución de Linux Ubuntu 14.04, ya que se ha desarrollado una aplicación GUI (Graphical User Interface) para que el usuario sea capaz de interactuar con el software. Esta aplicación grafica se ha desarrollado en el lenguaje de alto nivel Python, sobre el entorno de desarrollo Spyder.

Esta aplicación se ha enfocado a la creación de un registrador de datos o *Datalogger* programable, con mucha capacidad de almacenamiento y con la capacidad de envío remoto de estos datos mediante conexión a red, incluyendo el envío por email y la gestión de copias de seguridad. Además, los datos también pueden ser extraídos de forma local mediante un dispositivo de almacenamiento con interfaz USB. Los datos se almacenarán en formato CSV, ya que puede ser interpretado por cualquier programa de cálculo, como Microsoft Excel, Libre Office Calc, Matlab, etc. El *Datalogger* dispone de un servicio de salvaguarda de la información registrada mediante el envío de emails a un correo electrónico establecido por el usuario y la notificación de alarmas en caso de que se haya sobrepasado los límites de los canales de registro.

Se ha orientado inicialmente a la adquisición de datos de una estación meteorológica, para el registro de variables como: temperatura, humedad relativa y cantidad de lluvia. El *Datalogger* se puede dejar programado y que registre los datos sin la supervisión del usuario. Con estos datos se pueden conocer las condiciones meteorológicas de ese lugar, crear información climática, establecer predicciones, enviar los datos al Instituto Nacional de Estadística, etc.

ABSTRACT

This Final Degree Project is based on the Hardware / Software co-design of a PSoC (Programmable System on Chip), formed by a processing system (PS) based on a dual-core ARM Cortex-A9 microprocessor, running a Linux operating system, and the associated programmable logic (PL), all implemented on a device of the Zynq-7000 family of Xilinx. Both parties must interact with each other, but they are well differentiated. The logical part is programmed in VHDL language and follows the structure of data acquisition, sending this data to a FIFO buffer and later to the processing system. The PS part is based on the microprocessor system running a Xillinux device, which allows easy handling of all the interfaces of the ZedBoard card, such as VGA, Ethernet, USB, etc.

It has been developed on the ZedBoard board that incorporates the device XC7Z020-CLG484-1, of the Zynq-7000 family of Xilinx. In addition, it incorporates a license from the Vivado development environment, on which the hardware part will be worked. However, the software part has been developed in the distribution of Linux Ubuntu 14.04, since a GUI application (Graphical User Interface) has been developed so that the user is able to interact with the software. This graphic application has been developed in the high-level Python language, on the Spyder development environment.

This application has focused on the creation of a *Datalogger*, with a lot of storage capacity and with the ability to send this data remotely via a network connection, including sending by email and managing backup copies. In addition, the data can also be extracted locally using a storage device with USB interface. The data will be stored in CSV format, since it can be interpreted by any calculation program, such as Microsoft Excel, Libre Office Calc, Matlab, etc. The *Datalogger* has a service for safeguarding the registered information by sending emails to an email established by the user and notification of alarms in case the limits of the registration channels have been exceeded.

It has been initially oriented to the acquisition of data from a meteorological station, for the registration of variables such as: temperature, relative humidity and amount of rain. The *Datalogger* can be left programmed and that registers the data without the supervision of the user. With these data you can know the meteorological conditions of that place, create climate information, establish predictions, send these data to the National Statistics Institute, etc.

AGRADECIMIENTOS

En primer lugar, agradecer a Javier Vicuña, mi tutor de este TFG, todo el esfuerzo y las horas invertidas conmigo. Sin él nada hubiera sido posible.

Agradecer a mi familia el apoyo incondicional y la paciencia que han tenido conmigo durante estos duros cuatro años. Gracias por estar ahí incluso en los peores momentos y por todo lo que hacéis por mí. Además, agradeceros que me hayáis dado la oportunidad de haber estudiado una carrera, a pesar del esfuerzo que os supone.

A mi novia Aroa, por soportar mis cambios durante estos años. Por llamarme cada noche y conseguir que, aunque haya tenido un mal día, se convierta en bueno. Gran parte del apoyo me lo has dado tú.

A mis amigos de clase. Gracias por aportar vuestro granito de arena con vuestra amistad. Porque todos los buenos recuerdos que me llevo de la universidad son gracias a vosotros.

A ti yayo, porque sin ti no hubiera tenido fuerzas para acabar. Gracias por enseñarme a ser como soy. Gracias por darnos fuerza y cuidarnos desde arriba. Espero que te sientas orgulloso de tu nieto porque este TFG va para ti.

UNIVERSIDAD DE LA RIOJA

Trabajo Fin de Grado

Desarrollo de un “Datalogger” basado en ARM Cortex A9 y Linux mediante dispositivo Zynq-7000

Autor: Alberto Otaño Jiménez

Director: Javier Esteban Vicuña Martínez

Julio 2019

UNIVERSIDAD DE LA RIOJA

Trabajo Fin de Grado

Desarrollo de un “Datalogger” basado en ARM Cortex A9 y Linux mediante dispositivo Zynq-7000

1. ÍNDICE

Alberto Otaño Jiménez

Julio 2019

ÍNDICE GENERAL

MEMORIA

ÍNDICE DE FIGURAS

MEMORIA

ANEXOS

PLIEGO DE CONDICIONES

ÍNDICE DE TABLAS

MEMORIA

ANEXOS

PLIEGO DE CONDICIONES

UNIVERSIDAD DE LA RIOJA

Trabajo Fin de Grado

Desarrollo de un “Datalogger” basado en ARM Cortex A9 y Linux mediante dispositivo Zynq-7000

2. MEMORIA

Alberto Otaño Jiménez

Julio 2019

MEMORIA

1. PRESENTACIÓN

Este trabajo ha sido realizado por Alberto Otaño Jiménez, estudiante de cuarto curso del Grado en Ingeniería Electrónica Industrial y Automática y se presenta como Trabajo Fin de Grado al objeto de su evaluación académica para la obtención del título de Graduado por la Universidad de La Rioja.

2. OBJETO

2.1. Objetivo General

El objeto principal del trabajo ha consistido en el diseño y desarrollo de un sistema registrador de datos desatendido (en inglés *Datalogger*), basado en un microprocesador de 32bits embebido en un dispositivo programable Zynq-7000 del fabricante Xilinx, asociado a una tarjeta de desarrollo ZedBoard. El Sistema on Chip desarrollado, (en adelante SoC), cargará desde una tarjeta SD y arrancará un sistema operativo basado en una distribución Linux.

El desarrollo contempla dos bloques o partes bien diferenciadas: la parte hardware, que incluye tanto la captura de datos por parte del conversor analógico-digital integrado y disponible en el dispositivo programable Zynq-7000, como su accesibilidad para el sistema operativo; y la parte software, que incluye el desarrollo de una aplicación con interfaz gráfico de usuario (GUI) para configurar la captura de los datos, su monitorización y la exportación o gestión de dichos datos.

Como parte del objetivo principal, se propone la puesta a prueba del sistema desarrollado con una aplicación de estación meteorológica que capturará tres variables analógicas proveniente de tres sensores, con una determinada cadencia de muestreo. Los datos capturados se formatearán en un fichero CSV para que pueda ser fácilmente exportado o interpretado por un programa externo como Excel, Matlab, etc. El sistema prototipo *Datalogge*r desarrollado podrá capturar y almacenar localmente un volumen de datos en función de la capacidad de memoria del dispositivo de almacenamiento. Estos datos podrán ser adquiridos por el usuario de forma remota, mediante el envío de emails por parte del sistema a fin de crear una copia de seguridad periódica.

Un ejemplo de aplicación en el que se puede utilizar este sistema es el de registrado de los datos que provienen de una estación meteorológica. Para probar esta aplicación, se han conectado tres señales analógicas que provienen de tres potenciómetros, con su correspondiente circuito analógico de adaptación, para que simulen el comportamiento de tres sensores de la estación, por ejemplo: temperatura, humedad y sensor de nivel de un pluviómetro.

2.2. Objetivos particulares

Para la consecución del objetivo general descrito en el apartado anterior se han tenido que ir desarrollando una serie de objetivos particulares o etapas. A continuación, se enumeran brevemente algunas de las más relevantes:

- Análisis completo de las características y recursos disponibles del dispositivo FPGA Zynq-7000 de Xilinx, así como su funcionamiento en aplicaciones reales.
- Análisis y experimentación general de la tarjeta ZedBoard donde se aloja la FPGA Zynq-7000 y sus modelos de configuración y especialmente, las características del interfaz con el convertidor ADC, el interfaz, de video VGA, puerto USB y la tarjeta SD.
- Lectura e interpretación de hojas de características o datasheets del fabricante relacionadas con los dispositivos que maneja este proyecto, especialmente relativos al XADC, jumpers de configuración, etc.
- Perfeccionamiento y ampliación de los conocimientos sobre el lenguaje de descripción de hardware VHDL y su uso en el entorno de desarrollo Vivado de Xilinx, mediante la creación de aplicaciones sencillas con el sistema operativo Xillinux.
- Creación de un diseño hardware para que el sistema funcione como sistema de adquisición de datos que permita el flujo de datos de la FPGA al Host.
- Aprendizaje del sistema operativo Linux, instalación tanto a un ordenador como a una tarjeta SD, instalación de librerías, dependencias, programas, etc.
- Aprendizaje del lenguaje de programación de alto nivel Python.
- Aprendizaje y creación de aplicaciones de interfaz gráfica de usuario (GUI) que permita la comunicación del usuario con el nivel Hardware de la FPGA.

3. ALCANCE

Este proyecto contempla todas las fases necesarias para la consecución de los objetivos propuestos. Incluye tanto el diseño, desarrollo y la implementación de la parte hardware y de la parte software necesarias para el desarrollo del sistema *Datalogger* propuesto en el dispositivo FPGA, mediante un SoC basado en un microprocesador de 32 bits, corriendo una distribución Linux.

En Linux incluye una aplicación para la interacción con el usuario mediante la cual se va a programar la temporalización del registro de datos y cada cuanto tiempo se pretende que se salvaguarde la información. Para realizar esta copia de seguridad el sistema cuenta con un servicio de envío de emails a una dirección de correo electrónico previamente establecida por el usuario en la que se adjuntaran los archivos CSV con los datos de los tres canales de adquisición y la fecha del envío del mail. Además, el usuario puede establecer umbrales, tanto inferior como superior, en los que el sistema enviara un email notificando que se ha superado dicho umbral.

Una valiosa funcionalidad de los *Dataloggers* consiste en el envío al usuario de las alarmas en modo remoto (vía SMS, email…) para que de esta forma tenga un mejor control del estado del sistema. Otra funcionalidad interesante es la posibilidad de que el usuario pueda establecer la configuración de captura de forma remota, desde otro PC o desde su teléfono móvil. Ambas funcionalidades han quedado fuera del alcance de este proyecto porque sobrepasan los objetivos planteados y necesariamente, requerirían prolongar el tiempo de desarrollo y por tanto retrasar el depósito y defensa del trabajo. Algunas de estas funcionalidades se proponen para trabajos futuros en el apartado 11. VÍAS DE CONTINUACIÓN

4. ANTECEDENTES

4.1. Definición de *Datalogger*

Un registrador de datos o *Datalogger* es un dispositivo electrónico que registra datos comúnmente referenciados al tiempo y/o en relación al espacio (su ubicación), procedentes de instrumentos y/o sensores propios o conectados externamente. Mide las señales eléctricas de sensores a una velocidad de muestreo establecida y predeterminada, procesa y almacena los datos en local, y/o dispone de recursos de comunicación para enviar los datos captados por los sensores a una base de datos remota.

Su principal ventaja es que tienen la función automática de tomar cada uno de los muestreos. Gracias a esto, no necesitan la intervención de un usuario que active la captura de datos. Un *Datalogger* puede tomar muestras mientras esté operativo y alimentado con tensión y tenga memoria interna suficiente de almacenamiento, cuando almacena en local.

Este dispositivo tiene un funcionamiento autónomo desde el momento que se encuentra configurado y activado, ya que comúnmente se dejan sin vigilancia en el lugar donde se quieren muestrear datos. Suelen disponer de una interfaz de usuario para programar cuándo y cómo se desea iniciar la adquisición, tratamiento y almacenamiento de datos y puede disponer de uno o varios canales de adquisición de datos.

El precio de los *Datalogger* puede ser muy variado, sin embargo, de acuerdo a los avances de la tecnología, es posible encontrarlos a un bajo costo. Un ejemplo de ello es que se pueden adquirir equipos pequeños con un precio completamente accesible, mientras que si requiere un registrador preciso, el cual cuente con características más completas, el precio será mayor.

Cabe destacar que los registradores portátiles pueden ser usados para cualquier tipo de aplicaciones, debido a ello cuentan con diferentes características que los hacen únicos para la aplicación concreta en la que se quieran utilizar. Es importante comentar que normalmente el usuario configura el tiempo de muestreo, cada cuando requiere que

el equipo tome la lectura, el número de muestras y su periodicidad, con valores en el rango desde 1s hasta 1 muestra por semana.

Comentar también que los registradores portátiles usualmente no necesitan de otras herramientas adicionales para poder realizar su función, al contrario que las tarjetas de adquisición que requieren de otro tipo de elementos como una computadora, la cual le servirá entre otras funciones, para procesar y/o almacenar los datos. Estás características de los *Datatalogger* los hace independientes y cuentan con una memoria propia y suficiente espacio para el registro de los datos.

Pueden medir días, semanas, meses, años sin ningún problema e incluso pueden llevar años registrando datos sin que necesiten ningún tipo de supervisión. Cabe resaltar que esto es posible gracias a la memoria no volátil de almacenamiento con la que cuentan, que no requiere de energía para poder mantener la información almacenada y esto en comparación a otros registradores los hace muy eficientes.

Debido al largo periodo en que los dispositivos pueden realizar un muestreo de datos, precisan que se les pueda asociar una marca de tiempo (en inglés *Timestamp*) con la fecha y la hora a cada uno de los registros de datos obtenidos. Es esencial, por tanto, que cuenten con un sistema de reloj/calendario en tiempo real, utilizando de forma práctica precisiones del orden del segundo, aunque el sistema podría utilizar varios ordenes de magnitud de precisión temporal.

Existe en el mercado una gran variedad de modelos, desde los más sencillos, generalmente mono canal, hasta otros, de tipo multicanal que además incorporan diversas funciones de alerta al usuario y control remoto. Además, algunos *Datalogger* tienen funciones de servidor web. En este caso, el usuario podrá consultar los datos de forma remota y desde un ordenador y por supuesto, *Dataloggers* específicos para aplicaciones IoT.

Otra característica de estos sistemas frente a los sistemas de adquisición basados en computador, es su bajo consumo de energía y su robustez para operar en condiciones ambientales exigentes de humedad y temperatura o incluso específicos para trabajar sumergidos, Estas son características difícilmente alcanzables con sistemas de adquisición/registrado basados en computador.

Con la realización de este proyecto se pretende realizar un *Datalogger* con una gran capacidad de almacenamiento de datos, limitada por el dispositivo externo conectado para que almacene. Además, puede ser multifunción, ya que los sensores se conectarán externamente por el usuario y dependiendo de lo que la aplicación requiera, pueden ser unos u otros. En este caso, se ha pensado en la colocación de sensores de: temperatura, humedad y un sensor de nivel para un pluviómetro. Se ha dispuesto de tres variables a capturar, pero se podría aumentar este número hasta diecisiete. La utilización última que se le pretende dar es su colocación en una estación meteorológica.

4.2. Uso del *Datalogger* en una estación meteorológica

Una estación meteorológica es un dispositivo que recoge los datos de distintas variables atmosféricas que son de interés para la meteorología.

La utilidad principal de una estación meteorológica es recoger y registrar datos meteorológicos, con los cuales se obtiene valiosa información que puede tener entre otras, las siguientes funcionalidades:

- Saber exactamente las condiciones meteorológicas de ese ese lugar.
- Aportar información para realizar los pronósticos meteorológicos de diferentes lugares.
- Crear información climática representativa del lugar en donde se toman los datos.
- Crear alertas especificas ante fenómenos meteorológicos que pudieran ser de interés.
- Correlacionar fenómenos meteorológicos con situaciones de riesgo, accidentes, destrucción de infraestructuras, etc.
- Información para la agricultura. La información de las condiciones meteorológicas son una información de gran valor para las explotaciones agrícolas que usan está información para tomar decisiones.

Dependiendo del sensor que se coloque, existen diferentes variables que una estación meteorológica debe tomar. En el caso de este proyecto, como la conexión de sensores se realizará de forma externa, se puede tomar cualquier variable analógica dependiendo del sensor utilizado. Las variables más comunes son:

- Temperatura en aire
- Humedad
- Presión barométrica
- Velocidad del viento
- Dirección del viento
- Precipitaciones
- Nivel de UV
- Grosor de Nieve
- Temperatura en suelo
- Humedad del suelo
- Radiación solar
- Análisis de contaminación
- Medición de horas luz
- Medición de la altura de las nubes

Algunos de los instrumentos que incorporan las estaciones son los que se muestran a continuación. Estos instrumentos utilizan un determinado transductor que es capaz de transformar una magnitud física en una variable eléctrica.

- Termómetro (mide la temperatura)
- Pluviómetro (mide la cantidad de precipitaciones)
- Higrómetro (mide la humedad relativa del aire y la temperatura del punto de rocío)
- Anemómetro (mide la velocidad del aire)
- Barómetro (mide la presión atmosférica)
- Geotermómetro (mide la temperatura del subsuelo. Es muy útil para agricultura)
- Piranómetro (mide la radiación solar)
- Cielómetro (mide la distancia del suelo a la base de la nube)
- Nivómetro (mide la profundidad de la nieve con respecto al suelo)
- Radiómetro (mide la radiación electromagnética)

Existen diferentes tipos de estaciones meteorológicas, dependiendo de las funcionalidades que ofrezca cada una. Una clasificación de las diferentes estaciones puede ser vista a continuación:

- Estaciones meteorológicas domésticas
- Estaciones meteorológicas con conexión a PC
- Estaciones meteorológicas con conexión a internet
- Estaciones meteorológicas portátiles
- Estaciones meteorológicas profesionales automáticas
- Estaciones meteorológicas para aficionados

Este proyecto pretende realizar una estación meteorológica que con conexión USB, para que se puedan extraer los datos mediante esta fuente y con conexión a internet para que pueda enviar los datos por correo electrónico de forma remota, a modo de salvaguardado, cuando el usuario así lo haya configurado. Además, va a ser una estación meteorológica portátil, ya que se puede montar y desmontar de forma muy fácil y rápida.

5. NORMAS Y REFERENCIAS

5.1. Disposiciones legales y normativa aplicada

Este proyecto, por tratarse de un desarrollo electrónico y por sus características, se rige bajo la legislación de Baja Tensión. Adicionalmente se citan ciertas normas relacionadas:

- Reglamento Electrotécnico de Baja Tensión
- Normativa IEEE (Instituto de Ingeniería Eléctrica y Electrónica)
- IEEE 1149.1-2013 - Estándar IEEE para puerto de acceso de prueba y arquitectura de exploración de límite (JTAG)

- UNE 21-302 "Vocabulario Electrotécnico"
- UNE 20-324 "Grados de protección de los envolventes del material eléctrico de baja tensión
- UNE 21-401 "Conductores eléctricos aislados"
- Normativa para obtención del sello CE, para ser comercializado en la unión europea: ensayos de Compatibilidad Electromagnética (CEM)
    - Real Decreto 186/2016, de 6 de mayo de 2016, por el que se regula la compatibilidad electromagnética de los equipos eléctricos y electrónicos
    - Directiva 2014/30/UE del Parlamento Europeo y del Consejo, de 26 de febrero de 2014, sobre la armonización de las legislaciones de los Estados miembros en materia de compatibilidad electromagnética.
- UNE-EN611000-4-3-1998 "Compatibilidad Electromagnética"
- UNE 20-531-73 "Series de colores nominales para resistencias y condensadores"

Algunas de las normativas adicionales relacionadas con este documento técnico son:

- UNE 157001:2014 "Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico"

5.2. Programas y aplicaciones de desarrollo utilizadas

- Vivado 2018.2: se trata de un entorno de desarrollo integrado para lenguaje de descripción de hardware HDL, producida por Xilinx y que permite sintetizar diseños, realizar análisis de tiempo y configurar el dispositivo de destino. Solo permite configurar dispositivos Xilinx.
- USB Image Tool: es un programa para crear y gestionar imágenes de memorias USB con el que se puede hacer y recuperar copias de seguridad. Se utiliza sobre todo para la creación de imágenes de sistemas operativos en dispositivos externos como USBs, tarjeta SD, etc.
- Linux Live USB Creator: es un programa para la creación de imágenes de sistemas operativos en dispositivos externos extraíbles, como, por ejemplo: USB o DVD. Es un software de código libre y abierto, incorpora la función de virtualizar Linux sobre Windows sin la necesidad de reiniciar el ordenador, soporta muchas distribuciones de Linux y permite reservar un cierto espacio de persistencia para que los cambios ejecutados en Linux se mantengan en el USB
- Qt Creator: es un IDE multi plataforma programado en C++, JavaScript y QML para el desarrollo de aplicaciones con Interfaces Gráficas de Usuario (GUI) con las bibliotecas Qt. Soporta los sistemas operativos: Windows XP y superiores, Mac OS X 10.4 o Linux 2.6
- Qt Designer: es la herramienta de Qt para diseñar y construir interfaces gráficas de usuario (GUI) con Qt Widgets. Puede crear y editar ventanas o cuadros de diálogo de una manera que el usuario ve lo que diseña y lo puede probar utilizando diferentes estilos y resoluciones. Los widgets y formularios creados

con Qt Designer se integran con el código programado, utilizando señales y slots para que pueda asignar fácilmente el comportamiento a elementos gráficos. Todas las propiedades establecidas en Qt Designer pueden cambiarse dinámicamente dentro del código.

- Spyder: es un entorno de programación escrito para Python. Ofrece una combinación de las funcionalidades de edición, análisis, depuración y creación de perfiles. Trae consigo las funcionalidades de: editor, consola Python, explorador de variables, perfilador (para aumentar el rendimiento del código) y depurador. Además, se cuenta con la visualización de la interfaz gráfica de usuario que el usuario ha creado.
- Microsoft Excel 2016: es una aplicación de hojas de cálculo que forma parte de la suite Microsoft Office. Es una aplicación utilizada en tareas financieras y contables, con fórmulas, gráficos y un lenguaje de programación. Es una aplicación que corre únicamente bajo el sistema operativo Windows. Además, es capaz de interpretar el formato CSV (comma-separated values).
- Libre Office Calc: se trata de una aplicación de hojas de cálculo, parecida a Microsoft Excel, que corre bajo el sistema operativo de Linux. Ofrece las mismas funcionalidades que Excel, a diferencia de unas pocas. Se trata de una aplicación de uso libre y gratuito, a diferencia de Microsoft Excel, que también permite la interpretación del formato CSV.
- GIMP: (GNU Image Manipulación Program) es un programa de edición de imágenes digitales, tanto dibujos como fotografías. Es un programa libre y gratuito. Tiene herramientas que se utilizan para el retoque y edición de imágenes, dibujo de formas libres, cambiar el tamaño, recortar, hacer fotomontajes, convertir a diferentes formatos de imagen y otras tareas más especializadas.
- PSIM: es una herramienta de simulación de circuitos eléctricos y electrónicos por ordenador. La programación resulta muy sencilla e intuitiva, ya que se hace por medio de una interfaz gráfica que permite dibujar los esquemas de los circuitos que se desea simular, disponiendo para ello de paletas de elementos que incluyen además de los generadores y las cargas todos los elementos de control necesarios.

5.3. Bibliografía y enlaces web

A continuación, se citan fuentes de información que han ayudado a la realización de este trabajo:

[1] Diseño de un electrocardiograma portátil en la FPGA Zynq-7000: https://idus.us.es/xmlui/bitstream/handle/11441/69386/TFG_Pedro%20Gutierrez%20Lora.pdf?sequence=1&isAllowed=y

[2] DESARROLLO DE UN SISTEMA DE CAPTURA DE DATOS Y PROGNOSIS PARA ESTABLECER AVISOS DE MANTENIMIENTO EN MAQUINAS CNCs BASADO EN FACTORY 4.0: https://pdfs.semanticscholar.org/bd7d/8aa5542b67af556c8b8e5fc3bf6d29b5b846.pdf

[3] ZedBoard_RevC.1_Schematic_preliminary: http://zedboard.com/misc/files/ZedBoard_RevC.1_Schematic_preliminary.pdf
[4] ZedBoard_RevC.1_MechDrawing_preliminary: http://zedboard.com/misc/files/ZedBoard_RevC.1_MechDrawing_preliminary.pdf
[5] ZedBoard_RevC.1_Gerbers_preliminary: http://zedboard.com/misc/files/ZedBoard_RevC.1_Gerbers_preliminary.pdf
[6] ZedBoard_RevC.1_BOM_120625_preliminary: http://zedboard.com/misc/files/ZedBoard_RevC.1_BOM_120625_preliminary.pdf
[7] ZedBoard_HW_UG_v2_2: http://www.zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf
[8] ZedBoard_HW_UG_v1_3: http://www.zedboard.org/sites/default/files/ZedBoard_HW_UG_v1_3.pdf
[9] xillybus_product_brief: http://xillybus.com/downloads/xillybus_product_brief.pdf
[10] xillybus_host_programming_guide_linux: http://xillybus.com/downloads/doc/xillybus_host_programming_guide_linux.pdf
[11] xillybus_getting_started_xilinx: http://xillybus.com/downloads/doc/xillybus_getting_started_xilinx.pdf
[12] ug960-7series-ams-trd-user-guide: https://www.xilinx.com/support/documentation/boards_and_kits/ams101/2013_3/ug960-7series-ams-trd-user-guide.pdf
[13] ug908-vivado-programming-debugging: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug908-vivado-programming-debugging.pdf
[14] ug835-vivado-tcl-commands: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug908-vivado-programming-debugging.pdf
[15] ug761_axi_reference_guide: https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf
[16] ug585-Zynq-7000-TRM: https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf
[17] ug480_7Series_XADC: https://www.xilinx.com/support/documentation/user_guides/ug480_7Series_XADC.pdf
[18] ug476_7Series_Transceivers: https://www.xilinx.com/support/documentation/user_guides/ug476_7Series_Transceivers.pdf
[19] The_Zynq_Book_ebook_3: https://is.muni.cz/el/1433/jaro2015/PV191/um/The_Zynq_Book_ebook.pdf
[20] XADC Wizard v3.0 LogiCORE IP Product Guide (PG091) – Xilinx: https://www.xilinx.com/support/documentation/ip_documentation/xadc_wiz/v3_0/pg091-xadc-wiz.pdf

[21] ZedBoard Getting Started Guide: http://zedboard.org/sites/default/files/documentations/GS-AES-Z7EV-7Z020-G-V7-1.pdf
[22] analog-mixed-signal-product-brief: https://www.xilinx.com/publications/prod_mktg/analog-mixed-signal-product-brief.pdf
[23] 5066-PB-AES-Z7EV-7Z020-G-V3c (1)_0: http://zedboard.org/sites/default/files/product_briefs/5066-PB-AES-Z7EV-7Z020-G-V3c%20%281%29_0.pdf
[24] Datasheet TL082: https://pdf1.alldatasheet.com/datasheet-pdf/view/25384/STMICROELECTRONICS/TL082.html

En esta sección se detallarán los enlaces de páginas web y videos de YouTube que se han ido consultando durante la realización del trabajo. Se han ordenado por fases de ejecución del mismo.

- **Zynq-7000**

https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html [Último acceso: 12/06/2019]

- **ZedBoard**

https://www.xilinx.com/about/legal.html#general [Último acceso: 12/06/2019]

https://forum.digilentinc.com/topic/4477-zedboard-xadc-xillybus-not-working-properly/ [Último acceso: 12/06/2019]

- **Vivado**

https://forums.xilinx.com/ [Último acceso: 12/06/2019]

- ***Datalogger***

https://www.jmi.com.mx/literatura/blog/item/11-data-logger-que-es.html [Último acceso: 12/06/2019]

https://www.finaltest.com.mx/product-p/art-4.htm [Último acceso: 12/06/2019]

- **Estación Meteorológica**

https://estaciondemeteorologia.com/que-es-una-estacion-meteorologica/ [Último acceso: 12/06/2019]

- **Xillinux**

http://xillybus.com/ipfactory/ [Último acceso: 12/06/2019]

http://xillybus.com/tutorials/device-tree-zynq-1 [Último acceso: 17/06/2019]

http://xillybus.com/ [Último acceso: 12/06/2019]

- **Ubuntu**

https://computerhoy.com/paso-a-paso/software/como-instalar-ubuntu-equipo-windows-35619 [Último acceso: 12/06/2019]

https://es.wikipedia.org/wiki/BIOS [Último acceso: 12/06/2019]

https://maslinux.es/ejecucion-de-comandos-y-scripts-al-inicio-en-gnu-linux/ [Último acceso: 25/06/2019]

http://linux.dokry.com/cmo-enviar-datos-a-un-serial-port-y-ver-alguna-respuesta.html [Último acceso: 25/06/2019]

- **PyQt**

https://doc.qt.io/qt-5/qtdesigner-manual.html [Último acceso: 12/06/2019]

https://www.youtube.com/watch?v=2ANNq-QauZI [Último acceso: 12/06/2019]

https://www.tutorialspoint.com/pyqt/pyqt_using_qt_designer.htm [Último acceso: 12/06/2019]

https://es.wikipedia.org/wiki/Qt_Creator [Último acceso: 12/06/2019]

https://medium.com/@hektorprofe/primeros-pasos-en-pyqt-5-y-qt-designer-programas-gr%C3%A1ficos-con-python-6161fba46060 [Último acceso: 12/06/2019]

- **Python**

https://python-para-impacientes.blogspot.com/ [Último acceso: 12/06/2019]

https://python-para-impacientes.blogspot.com/2015/12/tkinter-interfaces-graficas-en-python-i.html [Último acceso: 25/06/2019]

https://www.youtube.com/watch?v=DDrM0uHTgIA [Último acceso: 25/06/2019]

https://python-para-impacientes.blogspot.com/2014/08/graficos-en-ipython.html [Último acceso: 25/06/2019]

https://www.tutorialspoint.com/python/tk_checkbutton.htm[Último acceso: 25/06/2019]

https://www.youtube.com/watch?v=hky6aQw65lw [Último acceso: 25/06/2019]

https://www.lawebdelprogramador.com/foros/Python/1290402-Interrupcion-de-software-en-python.html [Último acceso: 25/06/2019]

https://www.prometec.net/comandos-at-gsm-gprs-gps/ [Último acceso: 25/06/2019]

https://www.todavianose.com/enviar-sms-con-python-y-la-libreria-pyserial/ [Último acceso: 25/06/2019]

http://es.tldp.org/COMO-INSFLUG/COMOs/Serie-Como/Serie-Como-3.html [Último acceso: 25/06/2019]

https://es.wikipedia.org/wiki/Tkinter [Último acceso: 25/06/2019]

https://tecnonucleous.com/2018/01/28/como-instalar-pip-para-python-en-windows-mac-y-linux/ [Último acceso: 25/06/2019]

https://www.youtube.com/watch?v=KfDJOgqR4VU [Último acceso: 25/06/2019]

http://www.pythondiario.com/2014/12/enviar-un-correo-con-python-smtplib.html [Último acceso: 25/06/2019]

- **Normativa**

https://standards.ieee.org/standard/index.html?utm_source=mm_link&utm_campaign=find&utm_medium=std&utm_term=find%20standards [Último acceso: 12/06/2019]

https://www.ieee.org/documents/ieee_annual_report_09_complete.pdf [Último acceso: 12/06/2019]

- **Otras fuentes**

https://es.wikipedia.org/wiki/Microsoft_Excel [Último acceso: 12/06/2019]

https://es.wikipedia.org/wiki/Firmware [Último acceso: 12/06/2019]

- **Descargas**

https://www.xilinx.com/support/download.html [Último acceso: 12/06/2019]

http://xillybus.com/ipfactory/ [Último acceso: 12/06/2019]

https://www.ubuntu.com/download/desktop/thankyou?country=MX&version=14.04.3&architecture=i386 [Último acceso: 12/06/2019]

https://www.virtualbox.org/wiki/Downloads [Último acceso: 12/06/2019]

https://www.linuxliveusb.com/download. [Último acceso: 12/06/2019]

https://www.qt.io/download. [Último acceso: 12/06/2019]

https://launchpad.net/ubuntu/trusty/amd64/pyqt4-dev-tools/4.10.4+dfsg-1ubuntu1, [Último acceso: 12/06/2019]

https://gimp.uptodown.com/ubuntu [Último acceso: 12/06/2019]

http://xillybus.com/xillinux/ [Último acceso: 27/06/2019]

6. DEFINICIONES Y ABREVIATURAS

- ADC: Conversor analógico-digital (Analog Digital Converter)
- AMS: Tecnología de señal mixta analógica (Analog Mixed Signal). Permite remplazar sistemas analógicos complejos por ADC sencillos.
- APU: Unidad de procesamiento acelerado (Accelerated Processing Unit)
- ARM: Arquitectura de instrucciones reducidas de 32 y 64 bits para microprocesadores
- AXI: Interfaz extensible avanzado (Advanced eXtensible Interfaz)
- CSV: Formato para representar datos en forma de tabla separados por comas (comma-separated values)
- *Datalogger*: Registrador de datos
- Datasheet: Hoja de características o ficha técnica que resume el funcionamiento y otras características de un componente.
- EMIO: Multiplexador extendido de entradas y salidas (Extended Multiplexer Input Output)
- Ethernet: Estandar de redes de área local.
- FAT32: Tabla de asignación de archivos de 32 bits (File Allocation Table)
- FIFO: Buffer de entrada y salida de datos. Primero en entrar, primero en salir (First In First Out).
- FMC: estándar que define los modulos de expansión de entrada/salida con conexión a la FPGA (FPGA Mezzanine Card)
- FPGA: Matriz de puertas lógicas programable (Field Programmable Gates Array)
- GUI: Interfaz gráfica de usuario (Grafic User Interfaz)
- HDL: Lenguaje de descripción de hardware (Hardware Descripction Language)
- IDE: Entorno de desarrollo integrado (Integred Development Environment)
- Interfaz ULPI: Estandar de interfaz para sistemas USB 2.0 de alta velocidad (UTMI Low Pin Interfaz)
- IoT: Internet de las cosas (Internet of the Things)

- IP Core: Modulo de propiedad intelectual (Intellectual Property Core)
- JTAG: Puerto de prueba de acceso Estandar (Join Test Access Group)
- MIO: Multiplexor de entradas salidas (Multiplexer Input Output)
- MSB/LSB: Bit más significativo (More Significant Bit) / Bit menos significativo (Less Significant Bit)
- Pin-out: Asignacion del patillaje que se utiliza para determinar la función de cada pin en un circuito integrado.
- Python: Lenguaje de programación interpretado de alto nivel
- RTL: lógica de resistencia transistor (Resistor Transistor Logic)
- RTOS: Sistema operativo en tiempo real (Real Time Operative System)
- SCI: Interfaz de llamadas al sistema (System Call Interface)
- SD: Dispositivo en formato de tarjeta de memoria para dispositivos portátiles (Secure Digital)
- S/H: Circuito de conversión de tensión analógica a digital de muestreo y retención (Sample and Hold).
- SoC: Sistema en chip (System on Chip)
- TCL: Lenguaje de herramientas de comando (Tool Command Language)
- USB: Bus universal en serie (Universal Serial Bus)
- VGA: Matriz de gráficos de video (Video Graphics Array)
- VHDL: lenguaje de descripción de hardware cuyo nombre proviene de la combinación de dos acrónimos VHSIC (Very High Speed Integrated Circuit) y HDL (Hardware Description Language)
- XADC: Conversor analógico digital (ADC) de la familia Xilinx.
- Xilinx: Compañía distribuidora de dispositivos lógicos programables.

7. REQUISITOS DE DISEÑO

Para la consecución del objetivo final del proyecto se debe contar con ciertos requisitos de diseño que deben contemplarse en la solución final propuesta. Estos requisitos se encuentran enumerados a continuación:

- El prototipo del proyecto debe contar con disponibilidad para colocar tres sensores: de temperatura, de humedad y de nivel (para medir la cantidad de agua en un pluviómetro).
- El sistema sobre el que se realiza el *Datalogger* debe ser conectado a una tensión de alimentación de 12 V.
- El dispositivo tiene que contar con salida VGA y con entrada USB, Ethernet y ranura para SD, además de las entradas de adquisición de datos de los sensores.
- La aplicación grafica de interfaz con el usuario debe ser sencilla e intuitiva de forma que se pueda programar el comienzo, el paro, la configuración de la programación temporal y el envío de datos a través de internet.
- Se deben visualizar una cierta cantidad de valores captados por los sensores, en la pantalla principal de la aplicación, para que el usuario se asegure de que los

datos tomados son correctos. Se ha optado por la visualización de cuatro datos máximo en cada uno de los canales de adquisición.

- La salida de los datos debe ser en formato CSV (Comma-Separated Value) de forma que una hoja de cálculo como puede ser Excel o Libre Office Calc lo pueda interpretar, dibujar gráficas, hacer datos estadísticos, etc. La salida de datos en CSV se muestra de la siguiente manera:

  Canal de captura; año; mes; día; hora; minuto; dato muestreado; unidades

- El dispositivo sobre el que se ha de realizar el *Datalogger* ha de ser un dispositivo portátil, de forma que se pueda montar y desmontar de forma rápida y sencilla, para que se pueda situar en otro lugar en el que se quieran tomar muestras de datos.
- El entorno de colocación del dispositivo no debe ser un entorno muy agresivo, con gases corrosivos ni muy altas temperaturas, ya que tanto los sensores como la FPGA se pueden deteriorar.
- Se debe contar con prestaciones de salvaguardado de información, para que un fallo en el sistema no se pierdan los datos obtenidos hasta la fecha. Para ello se ha de disponer de un sistema de envío remoto, vía email, mediante el cual el usuario establecerá un tiempo en el que desea que se le envié un email a un correo electrónico que previamente habrá establecido en la configuración.
- El usuario establecerá unos umbrales máximo y mínimo entre los que debe oscilar el rango de variables medidas. En caso de que se superen dichos umbrales, el sistema será capaz de enviar un email al correo electrónico establecido en configuración.
- El envío de alarmas por superar los umbrales de oscilación, puede ser enviado vía SMS. No es el caso de este proyecto, se propone en el apartado 11. VÍAS DE CONTINUACIÓN
- La configuración del sistema ha de realizarse de forma presencial junto al *Datalogger*. Otra opción sería la de realizarla de forma remota desde un dispositivo móvil o un PC conectados a internet. Se propone dicha posibilidad en el apartado 11. VÍAS DE CONTINUACIÓN

8. ANÁLISIS DE SOLUCIONES

8.1. Introducción teórica

8.1.1. FPGA Zynq-7000

8.1.1.1. Introducción a las características

La gama de dispositivos Zynq de Xilinx la forman dispositivos que combinan una FPGA y un procesador hardware, por lo tanto, sus características, capacidades y aplicaciones son diferentes a las de una FPGA o un procesador de forma aislada.

La familia Zynq-7000 SoC integra la capacidad de programación de software de un procesador basado en ARM con la capacidad de programación de hardware de una FPGA.

Los dispositivos Zynq-7000 están equipados con procesadores ARM Cortex-A9 de doble núcleo integrados con 28nm Artix-7 o lógica programable basada en Kintex-7 para un excelente rendimiento y la máxima flexibilidad de diseño. Los dispositivos Zynq-7000, con hasta 6.6M de células lógicas y transmisores (transceptores) que van desde 6.25Gb/s a 12.5Gb/s, lo que permiten diseños altamente diferenciados para una amplia gama de aplicaciones integradas, incluidos sistemas de asistencia con controladores de varias cámaras y 4K, 2K, Ultra-HDTV, etc. (Figura 1: Arquitectura Zynq-7000)

En Zynq, el ARM Cortex-A9 es un procesador de grado de aplicación, capaz de ejecutar sistemas operativos completos como Linux, mientras que la lógica programable se basa en la arquitectura FPGA de la serie 7 de Xilinx.

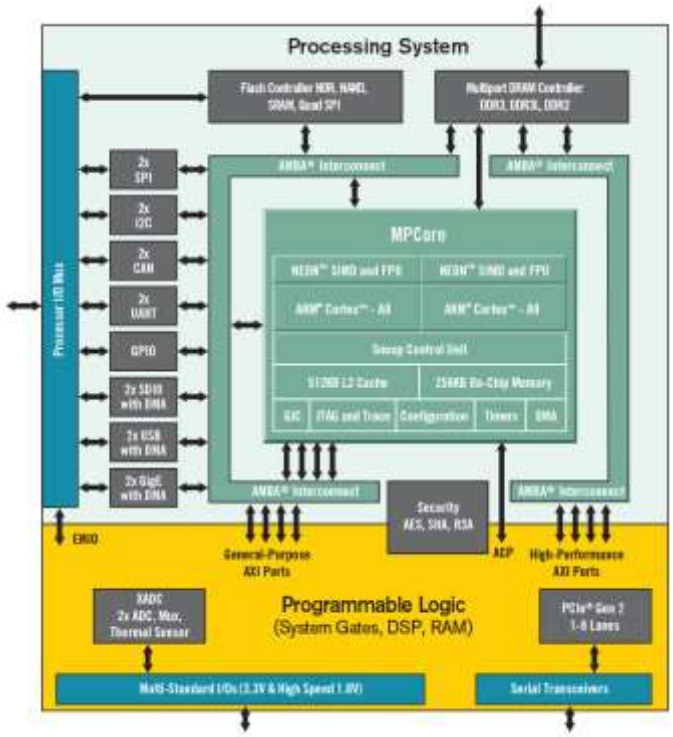


Figura 1: Arquitectura Zynq-7000

8.1.1.2. Características de Zynq-7000

8.1.1.2.1. Lógica programable (PL)

La parte PL del dispositivo Zynq se muestra en la Figura 2, con varias características resaltadas. El PL está compuesto predominantemente de tejido lógico de FPGA de propósito general, que se compone de bloques lógicos configurables (CLB), y

de bloques de entrada/salida (IOB) para la interfaz. A continuación, se explicará cada una de las características etiquetadas en la Figura 2.

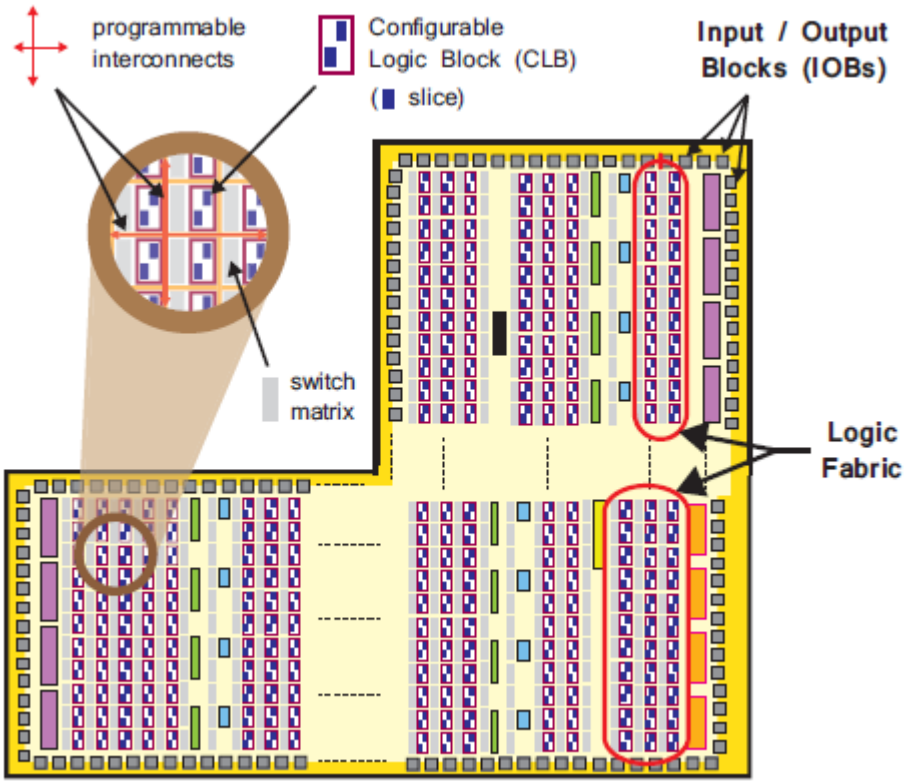


Figura 2:Lógica de la Zynq-7000

- **Bloque lógico configurable (CLB):** los CLB son agrupaciones pequeñas y regulares de elementos lógicos que se presentan en una matriz bidimensional en el PL y se conectan a otros recursos similares a través de interconexiones programables. Cada CLB se coloca al lado de una matriz de conmutación y contiene dos segmentos lógicos, como se muestra en la Figura 3.

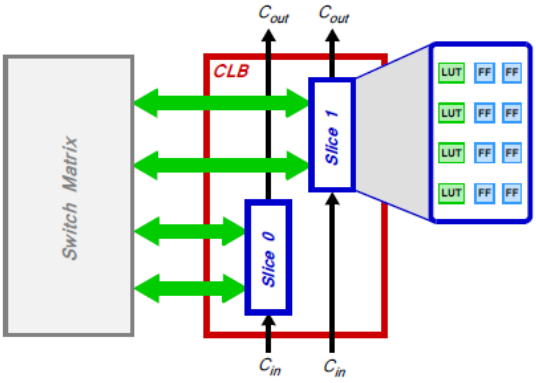


Figura 3: Composición de un CLB

- **Slice**: se trata de una subunidad dentro del CLB, que contiene recursos para implementar circuitos lógicos combinatorios y secuenciales. Como se indica en

la Figura 3, los segmentos de Zynq se componen de 4 tablas de búsqueda, 8 flip-flops y otra lógica.

- **Look Up Table (LUT):** un recurso flexible capaz de implementar una función lógica de hasta seis entradas, una pequeña memoria de solo lectura (ROM), una pequeña memoria de acceso aleatorio (RAM), o un registro de desplazamiento. Las LUT se pueden combinar para formar funciones lógicas, memorias o registros de desplazamiento más grandes.
- **Flip-flop (FF):** es un elemento de circuito secuencial que implementa un registro de 1 bit, con funcionalidad de restablecimiento.
- **Matriz de conmutación**: una matriz de conmutación se encuentra junto a cada CLB y proporciona una facilidad de enrutamiento flexible para realizar conexiones entre elementos dentro de un CLB y de un CLB a otros recursos dentro de la lógica programable.
- **Carry Logic**: los circuitos aritméticos requieren que las señales intermedias se propaguen entre segmentos adyacentes, y esto se logra mediante la denominada *Carry Logic* (Lógica de transporte). El *Carry Logic* comprende una cadena de rutas y multiplexores para vincular segmentos en una columna vertical.
- **Bloques de entrada/salida (IOB):** los IOB son recursos que proporcionan una interfaz entre los recursos lógicos PL y los circuitos externos del dispositivo físico (en el caso de este proyecto, la tarjeta ZedBoard). Cada IOB puede manejar una señal de entrada o salida de 1 bit. Las IOB se ubican generalmente alrededor del perímetro del dispositivo.

La lógica programable de Zynq-7000 tiene además recursos especiales como los que se muestran en la Figura 4: RAM de bloque para requisitos de memoria densa y cortes DSP48E1 para aritmética de alta velocidad. Ambos de estos recursos están integrados en la matriz lógica en una disposición de columnas, incrustados en la lógica de la estructura y normalmente cerca uno del otro.

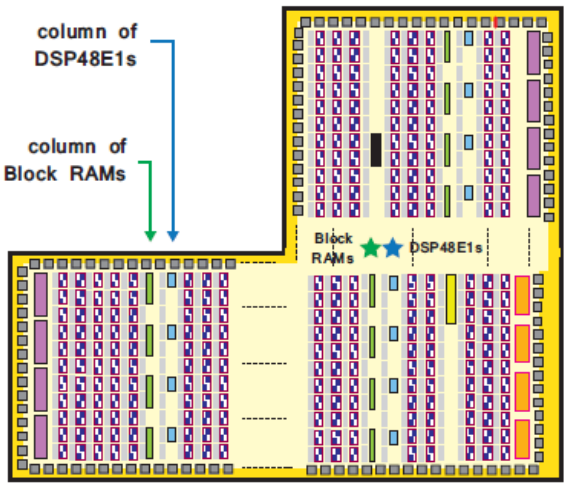


Figura 4:: Situación de RAMs y DSP48E1s dentro de la lógica programable

- **Las RAM:** pueden implementar memorias de acceso aleatorio (RAM), memoria de solo lectura (ROM) y memorias tipo FIFO. Cada RAM de bloque puede

almacenar hasta 36 Kb de información, y puede configurarse como una RAM de 36 Kb, o dos RAM de 18 KB independientes. El tamaño de palabra predeterminado es de 18 bits, y en esta configuración, cada RAM comprende 2048 elementos de memoria. La RAM también puede ser "remodelada" de manera que contenga más elementos más pequeños (por ejemplo: 4096 elementos x 9 bits, o 8192 x 4 bits), o alternativamente, menos elementos más largos (por ejemplo: 1024 elementos x 36 bits, 512 x 72 bits). Las memorias de mayor capacidad pueden formarse combinando dos o más RAM de bloque juntas.

- **Procesador digital de señal (DSP48E1):** Los DSP48E1 son segmentos especializados para implementar aritmética de alta velocidad en señales con longitudes de palabra aritméticas de media a larga. Comprenden principalmente un sumador (o restador), un multiplicador y un sumador (o restador) lógico, como se muestra en la Figura 5. donde se marcan las longitudes máximas de palabras aritméticas.

  Los DSP48E1 son adecuados para una variedad de aplicaciones en el procesamiento de señales, además de ser altamente eficientes y tener un alto rendimiento en cálculos de DSP.

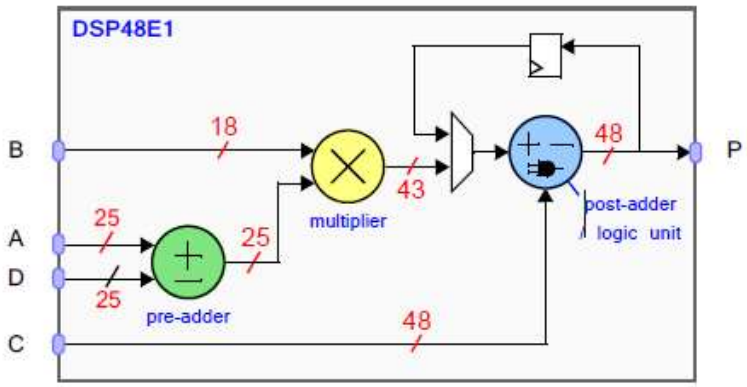


Figura 5: Procesador digital de señal (DSP48E1)

- Los recursos de entrada/salida (IOB) de propósito general en el Zynq están organizados en bancos de 50 IOB cada uno. Cada IOB contiene un ‘pad’, que proporciona la conexión física al mundo exterior.
- Los bancos de E/S están se dividen en Alto rendimiento (HP) o Alto rango (HR), y estos admiten varios voltajes:
- Las E/S de HP están limitadas a voltajes de 1,8V y se usan para interfaces de alta velocidad, para acceso a memoria
- Las interfaces de HR permiten voltajes de hasta 3.3V y permiten más variedad de estándares de E/S.

Los dispositivos Zynq incluyen transceptores GTX que se tratan de bloques de interfaz de comunicaciones de alta velocidad que están integrados en el tejido lógico. Estos son capaces de admitir una serie de interfaces estándar como, por ejemplo: PCI Express, Serial RapidIO, SCSI y SATA.

Los transceptores GTX se implementan como grupos de 4 canales individuales, cada uno de los cuales incluye un PLL (phase-locked loop) dedicado para ese canal, un transmisor y un receptor. Se admiten velocidades de hasta 12.5Gbps. Las interfaces se pueden usar para crear conexiones a dispositivos externos independientes, como equipos de red, discos duros y otras FPGAs.

También existen otras interfaces externas a la lógica programable como son:

- **Conversión analógica a digital (ADC):** la lógica programable incluye otro bloque como es el XADC, que cuenta con dos ADC de 12 bits separados, capaces de muestrear señales de entrada analógicas externas a 1Msps (mega muestra por segundo). Dado que se trata de un bloque principal en el proyecto, se estudiará en detalle en apartados posteriores.
- **Clocks**: la PL recibe cuatro entradas de reloj (separadas del sistema) y, además, es capaz de generar y distribuir sus propias señales de reloj independientes del sistema.
- **Programación y debug:** se proporciona un conjunto de puertos JTAG (Joint Test Action Group) en la sección PL para facilitar la configuración y la depuración de la lógica programable. Aunque normalmente se prefieren métodos más seguros en la implementación, la configuración JTAG se usa a menudo durante el desarrollo. Las instalaciones ofrecidas a través de JTAG admiten el debuging con herramientas de ARM y Xilinx.

8.1.1.2.2. Sistema de procesamiento (PS)

Todos los dispositivos Zynq tienen la misma arquitectura básica que contiene, como base del sistema de procesamiento, un procesador “hard” ARM Cortex-A9 de doble núcleo, ya incorporado. La alternativa a un procesador “hard” es un procesador "soft" como Xilinx MicroBlaze, que se forma combinando elementos de la estructura lógica programable (PL). La implementación de un procesador de software es, por lo tanto, el equivalente a cualquier otro bloque de IP implementado en la estructura lógica de una FPGA. Los procesadores duros pueden lograr un rendimiento considerablemente mayor, como ocurre con el procesador ARM de Zynq.

Hay que señalar que uno o más procesadores “soft” MicroBlaze se pueden usar dentro de la parte PL del Zynq (Figura 6), para operar de forma conjunta con el procesador ARM. MicroBlaze puede tener, por ejemplo, la misión de coordinar funciones específicas de bajo nivel dentro del sistema (tareas menos exigentes que pueden delegarse fuera del procesador principal Cortex-A9) para mejorar el rendimiento general.

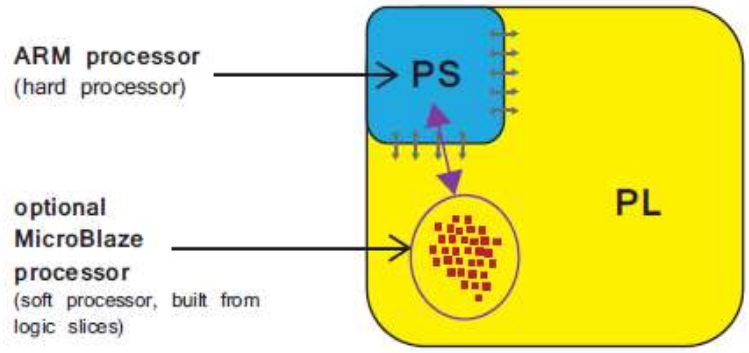


Figura 6: Coexistencia de parte PL y PS

El sistema de procesamiento Zynq no solo comprende el procesador ARM, sino un conjunto de recursos de procesamiento asociados que forman una Unidad de Procesamiento de Aplicaciones (en lo sucesivo APU) y otras interfaces periféricas: memoria caché, interfaces de memoria, interconexión y circuitos de generación de Clock. Un diagrama de bloques que muestra la arquitectura del PS se muestra en la Figura 7, donde se resalta la APU.

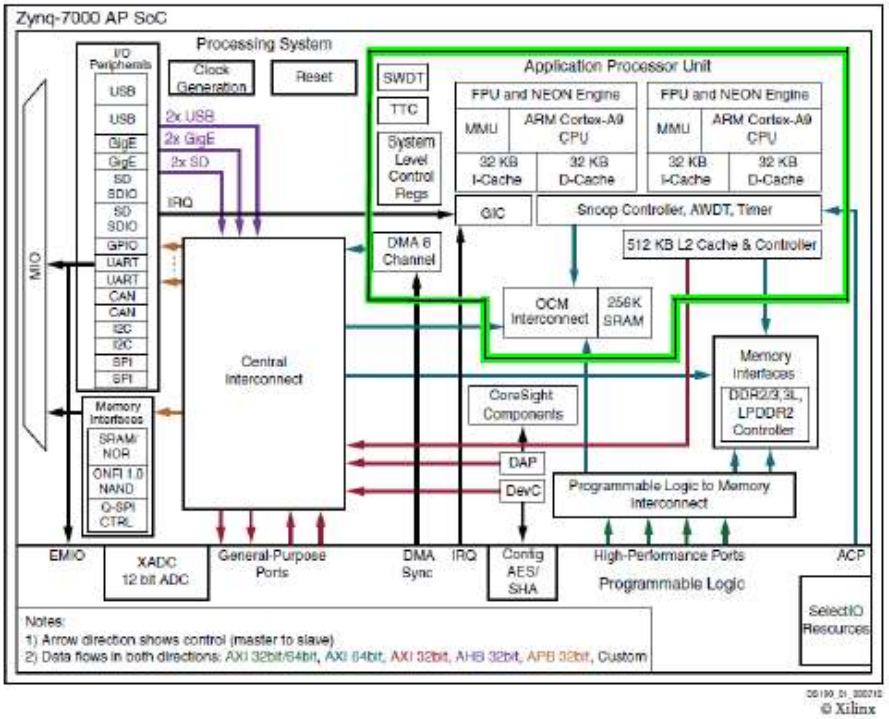


Figura 7: Diagrama de bloques de PS

En la Figura 8 se muestra un diagrama de bloques simplificado de la APU. La APU se compone principalmente de dos núcleos de procesamiento ARM, cada uno con unidades computacionales asociadas: un motor de procesamiento de medios (MPE) NEON y una unidad de punto flotante (FPU), una Unidad de Gestión de la Memoria (MMU), y una memoria caché de nivel 1 (L1 en dos secciones para instrucciones y datos). La APU también contiene una memoria caché de nivel 2 (L2) y una memoria de

chip adicional (OCM). Finalmente, una unidad de control Snoop (SCU) forma un puente entre los núcleos ARM, la memoria caché de nivel 2 y las memorias OCM.

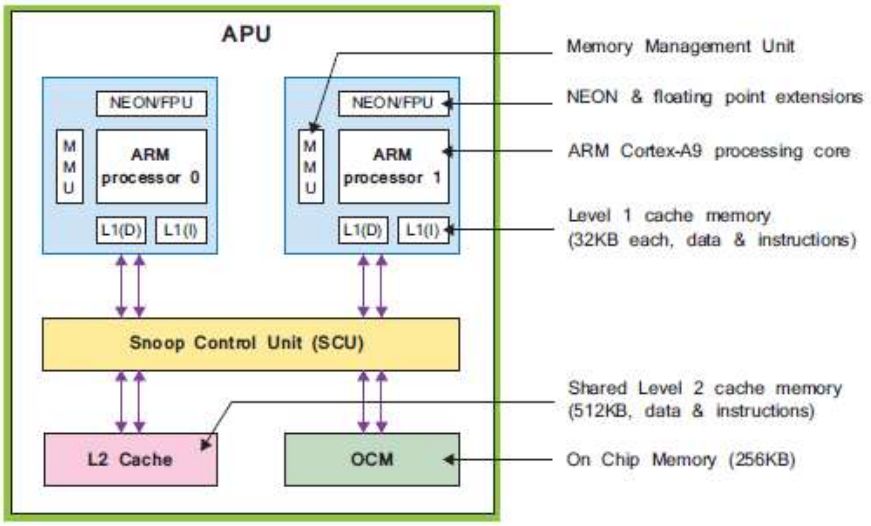


Figura 8: Diagrama simplificado de la APU

El ARM Cortex-A9 puede funcionar a hasta 1 GHz. Cada uno de los dos núcleos tiene cachés de Nivel 1 separados para datos e instrucciones, ambos de los cuales son de 32 KBytes. Esto permite el almacenamiento local de datos e instrucciones requeridos con frecuencia para tiempos de acceso rápidos y rendimiento óptimo del procesador. Los dos núcleos además comparten una memoria caché de nivel 2 de 512 KBytes para instrucciones y datos, y hay otros 256 KBytes de memoria en chip (OCM) dentro de la APU.

El Zynq PS presenta una gran variedad de interfaces entre el PS (Processing System) y el PL (Programmable Logic) como con los componentes externos, como se muestra en la Figura 7.

La comunicación entre la PS y las interfaces externas se logra principalmente a través de la E/S multiplexada (MIO) que permite que se pueda definir a gusto del usuario la asignación entre periféricos y pines. Existen varias conexiones que se pueden realizar a través del MIO extendido (EMIO), que no es una ruta directa entre el PS y las conexiones externas, sino que pasa a través de E/S de la PL. Estas conexiones se muestran en el lado izquierdo de la Figura 7. El EMIO se puede usar cuando se requiere una extensión más allá de los 54 pines disponibles del MIO o como un método de interfaz del PS con un bloque de IP implementado en el PL.

Las interfaces periféricas de entrada/salida se encuentra definido junto a sus características en la siguiente Tabla 1.

Interfaz E/S	**Descripción**
2xSPI	Serial Peripheral Interface: estándar para comunicaciones en serie

	basadas en una interfaz de 4 pines. Se puede utilizar en modo maestro o esclavo.
2xI2C	Inter Integrated Circuit: admite los modos maestro y de esclavo.
2x Bus CAN	Controller Area Network: controlador de interfaz de bus compatible con las normas ISO 118980-1, CAN 2.0A y CAN 2.0B.
2x UART	Universal Asynchronous Receiver-Transmitter: interfaz para datos de baja velocidad en comunicación en serie. Se suele utilizar para conexiones de terminal a una PC host.
GPIO (4x32)	General Purpose Input/Output: hay 4 bancos GPIO, cada uno de 32 bits.
2xSD	Para la interfaz con la memoria SD de la tarjeta ZedBoard.
2xUSB	Universal Serial Bus: compatible con USB 2.0 y se puede usar como host, dispositivo o de manera flexible.
2xGigE	Ethernet: compatible con velocidades de 10Mbps, 100Mbps y 1Gbps.

Tabla 1: Interfaces periféricas de la PS con el exterior

8.1.1.2.3. Conexión entre PS y PL

La clave del buen funcionamiento y rendimiento de la Zynq-7000 es que no solo cuenta con la posibilidad de tener una lógica programable y un sistema de procesamiento, sino que pueden ser conectados para aumentar su rendimiento. Para ello utiliza, sobre todo, las interfaces AXI y EMIO.

AXI (Advanced Extensible Interfaz o Interfaz Extensible Avanzada) forma parte del estándar abierto ARM AMBA (Advanced Microcontroller Bus Architecture o Arquitectura avanzada de Bus para Microcontroladores). Hay tres tipos de interfaces AXI4:

- AXI4: para requisitos de memoria de alto rendimiento asignados.
- AXI4-Lite: para una comunicación simple, de bajo rendimiento, asignada en memoria
- AXI4-Stream: para transmisión de datos a alta velocidad

AXI4 proporciona mejoras en la Zynq-7000, sobre todo en términos de productividad, flexibilidad y disponibilidad:

- Productividad: al estandarizar la interfaz AXI, los desarrolladores solo necesitan aprender un protocolo único para IP.
- Flexibilidad: proporcionar el protocolo correcto para la aplicación
- Disponibilidad: al pasar a un estándar de la industria, tiene acceso no solo al catálogo IP de Xilinx, sino también a una comunidad mundial de socios ARM.

- Muchos proveedores de IP soportan el protocolo AXI.
- Se encuentra disponible una amplia colección de proveedores de herramientas AXI.

AXI describe una comunicación entre un solo maestro y un solo esclavo, que representa núcleos IP que intercambian información entre sí. Los maestros y esclavos AXI pueden conectarse entre sí mediante una estructura llamada bloque de Interconexión. Las interfaces AXI4 y AXI4-Lite constan de cinco canales diferentes: leer canal de dirección, escribir canal de dirección, leer canal de datos, escribir canal de datos, escribir canal de respuesta.

Los datos pueden moverse en ambas direcciones entre el maestro y el esclavo simultáneamente, y los tamaños de transferencia pueden variar. El límite en AXI4 es una transacción de hasta 256 transferencias de datos. Sin embargo, AXI4-Lite permite solo 1 transferencia de datos por transacción (Figura 9)

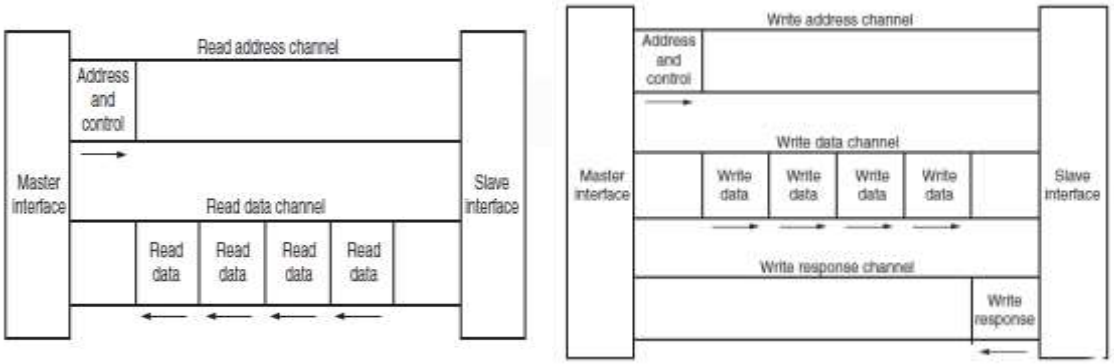


Figura 9:: A la izquierda: canal de lectura AXI. A la derecha: canal de escritura AXI

EMIO (Extended Multipurpose Input/Output) implica la transferencia de señal entre PS y PL. Las conexiones están dispuestas en dos bancos de 32 bits (Figura 10).

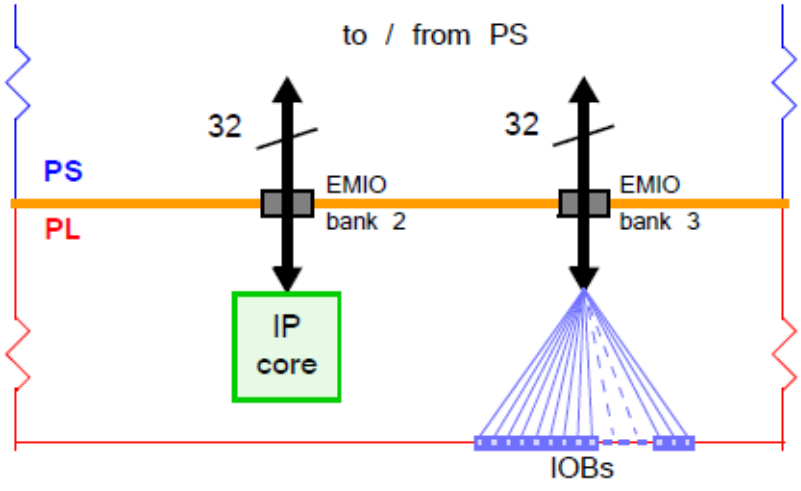


Figura 10: conexión PS-PL a través de EMIO

Las interfaces a través del EMIO en muchos casos están conectadas directamente a los pines externos de la PL. De este modo, EMIO puede proporcionar 64 entradas adicionales y 64 salidas. Otra opción es usar EMIO para interconectar el PS con un bloque periférico en el PL.

Dentro de la familia Zynq-7000 existen varios dispositivos de FPGAs. Para este proyecto se ha utilizado el dispositivo XC7Z020-CLG484-1. A continuación se muestra un resumen de las características de toda la familia Zynq-7000 (Figura 11 y Figura 12).

	Device Name	Z-7007S	Z-7012S	Z-7014S	Z-7010	Z-7015	Z-7020	Z-7030	Z-7035	Z-7045	Z-7100
	Part Number	XC7Z007S	XC7Z012S	XC7Z014S	XC7Z010	XC7Z015	XC7Z020	XC7Z030	XC7Z035	XC7Z045	XC7Z100
Processing System	Processor Core	Single-core ARM Cortex-A9 MPCore™ with CoreSight™			Dual-core ARM Cortex-A9 MPCore™ with CoreSight™						
	Processor Extensions	NEON™ & Single / Double Precision Floating Point for each processor									
	Maximum Frequency	667 MHz (-1); 766 MHz (-2)			667 MHz (-1); 766 MHz (-2); 866 MHz (-3)			667 MHz (-1); 800 MHz (-2); 1 GHz (-3)			667 MHz (-1) 800 MHz (-2)
	L1 Cache	32 KB Instruction, 32 KB data per processor									
	L2 Cache	512 KB									
	On-Chip Memory	256 KB									
	External Memory Support[1]	DDR3, DDR3L, DDR2, LPDDR2									
	External Static Memory Support[1]	2x Quad-SPI, NAND, NOR									
	DMA Channels	8 (4 dedicated to Programmable Logic)									
	Peripherals[1]	2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO									
	Peripherals w/ built-in DMA[1]	2x USB 2.0 (OTG), 2x Tri-mode Gigabit Ethernet, 2x SD/SDIO									
	Security[2]	RSA Authentication, and AES and SHA 256-bit Decryption and Authentication for Secure Boot									
	Processing System to Programmable Logic Interface Ports (Primary Interfaces & Interrupts Only)	2x AXI 32b Master 2x AXI 32-bit Slave 4x AXI 64-bit/32-bit Memory AXI 64-bit ACP 16 Interrupts									

Figura 11: Tabla comparativa del PS de la familia Zynq-7000

	Device Name	Z-7007S	Z-7012S	Z-7014S	Z-7010	Z-7015	Z-7020	Z-7030	Z-7035	Z-7045	Z-7100
	Part Number	XC7Z007S	XC7Z012S	XC7Z014S	XC7Z010	XC7Z015	XC7Z020	XC7Z030	XC7Z035	XC7Z045	XC7Z100
Programmable Logic	Xilinx 7 Series Programmable Logic Equivalent	Artix®-7 FPGA	Artix-7 FPGA	Artix-7 FPGA	Artix-7 FPGA	Artix-7 FPGA	Artix-7 FPGA	Kintex®-7 FPGA	Kintex-7 FPGA	Kintex-7 FPGA	Kintex-7 FPGA
	Programmable Logic Cells	23K	55K	65K	28K	74K	85K	125K	275K	350K	444K
	Look-Up Tables (LUTs)	14,400	34,400	40,600	17,600	46,200	53,200	78,600	171,900	218,600	277,400
	Flip-Flops	28,800	68,800	81,200	35,200	92,400	106,400	157,200	343,800	437,200	554,800
	Block RAM (# 36 Kb Blocks)	1.8 Mb (50)	2.5 Mb (72)	3.8 Mb (107)	2.1 Mb (60)	3.3 Mb (95)	4.9 Mb (140)	9.3 Mb (265)	17.6 Mb (500)	19.2 Mb (545)	26.5 Mb (755)
	DSP Slices (18x25 MACCs)	66	120	170	80	160	220	400	900	900	2,020
	Peak DSP Performance (Symmetric FIR)	73 GMACs	131 GMACs	187 GMACs	100 GMACs	200 GMACs	276 GMACs	593 GMACs	1,334 GMACs	1,334 GMACs	2,622 GMACs
	PCI Express (Root Complex or Endpoint)[3]		Gen2 x4			Gen2 x4		Gen2 x4	Gen2 x8	Gen2 x8	Gen2 x8
	Analog Mixed Signal (AMS) / XADC	2x 12 bit, MSPS ADCs with up to 17 Differential Inputs									
	Security[2]	AES and SHA 256b for Boot Code and Programmable Logic Configuration, Decryption, and Authentication									

Figura 12: Tabla comparativa del PL de la familia Zynq-7000

8.1.1.3. Sistemas Operativos (Linux) sobre Zynq-7000

Aunque un sistema operativo incorporado no es necesario en todas las aplicaciones de sistemas embebidos, su uso tiene varias ventajas que se enumeran a continuación:

- Reduce el tiempo de desarrollo: Al desarrollar un sistema integrado, hay una serie de áreas clave en las que un sistema operativo incorporado puede reducir el tiempo de desarrollo.
- Hace uso de las características existentes: Los sistemas operativos integrados ofrecen compatibilidad con muchas características que, de lo contrario, tendrían que ser controladas por el sistema, como, por ejemplo, al incorporar una pantalla el sistema debe poder admitirla y para ello es necesario desarrollar IPs que la controlen correctamente, tanto a nivel de controlador como a nivel de interfaz gráfica. Sin embargo, utilizando un sistema operativo se podrá tener acceso a todas las características que ofrece.
- Reduce los costes de mantenimiento y desarrollo: al hacer uso de un sistema operativo incorporado, se reduce la cantidad de código personalizado que debe desarrollarse, lo que equivale a un número mucho menor de errores que se pueden introducir en el software, lo que a su vez reduce el tiempo de probar el sistema para encontrarlos y eliminarlos. Un sistema operativo proporcionará una plataforma estable, para que todos los esfuerzos se concentren en desarrollar las aplicaciones personalizadas y no en depurar el código de bajo nivel.

La Figura 13 representa una arquitectura generalizada de alto nivel de un sistema GNU/Linux. En ella se observa como el kernel comunica la parte hardware y software. El kernel constituye una parte fundamental del sistema operativo, y es el principal responsable de facilitar a los distintos programas acceso seguro al hardware de la computadora. Es decir, es el encargado de gestionar recursos, a través de servicios de llamada al sistema. Como hay muchos programas y el acceso al hardware es limitado, también se encarga de decidir qué programa podrá usar un dispositivo de hardware y durante cuánto tiempo, lo que se conoce como multiprogramación. Acceder al hardware directamente puede ser realmente complejo, por lo que los núcleos suelen implementar una serie de abstracciones del hardware. Esto permite esconder la complejidad, y proporcionar una interfaz limpia y uniforme al hardware, lo que facilita su uso al programador. En la Figura 14 se muestra las diferentes llamadas al sistema que gestiona el kernel y sus conexiones con el hardware.

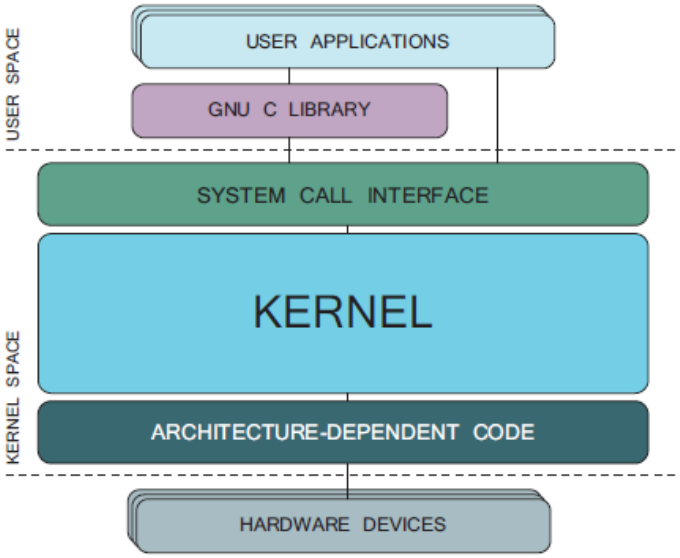


Figura 13: Arquitectura de un sistema GNU/Linux

Las aplicaciones y los programas del sistema, así como la biblioteca GNU C, forman parte del espacio de usuario, y se encuentran en la parte superior del kernel. Estas aplicaciones realizan llamadas al sistema y pueden requerir incluso recursos hardware. Es el kernel el que se encarga de organizar toda esta información y comunicación con la parte hardware

La Interfaz de llamadas del sistema (SCI) facilita las llamadas de función desde el espacio del usuario al núcleo del sistema. El kernel de Linux constituye el corazón del sistema operativo y proporciona un conjunto de herramientas con las cuales el espacio del usuario puede interactuar con el hardware. El propio kernel se puede dividir sus propios subsistemas en capas, como la gestión de procesos y memoria, el sistema de archivos virtual y los controladores de dispositivos (Figura 14).

El hardware físico reside en el extremo opuesto de la cadena y se comunica con el nivel de usuario mediante el kernel.

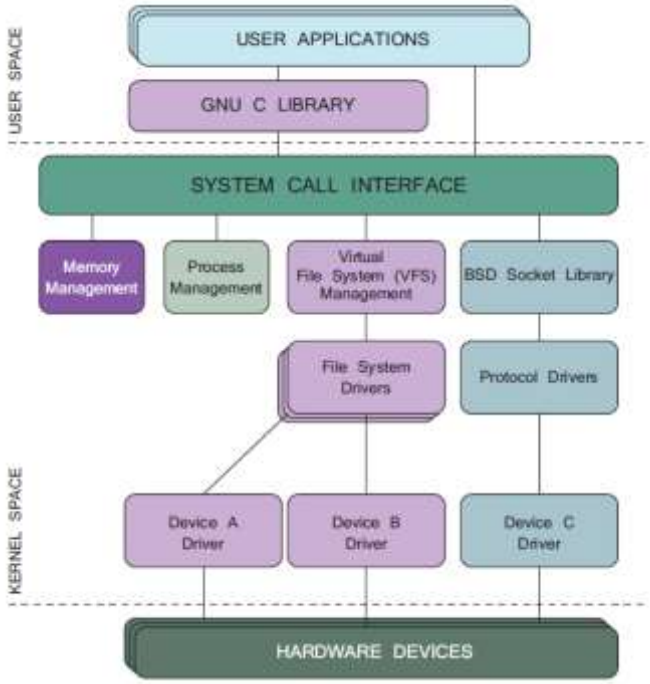


Figura 14:: Estructura del kernel de Linux

8.1.2. ZedBoard

8.1.2.1. Introducción

ZedBoard es una placa de evaluación y desarrollo basada en el dispositivo programable de Xilinx Zynq-7000. Combinando un sistema de procesamiento (PS) dual Cortex-A9 con 85000 celdas de lógica programable PL, el SoC (System on Chip) de Zynq-7000 puede enfocarse para un uso amplio en muchas aplicaciones. La combinación robusta de ZedBoard de periféricos a bordo y capacidades de expansión lo convierten en una plataforma ideal para el desarrollo de aplicaciones. Además, es una placa de bajo coste para Zynq-7000 y contiene todo lo necesario para montar un sistema Linux, Android, Windows u otro diseño basado en un sistema operativo.

ZedBoard incluye Xilinx XADC, FMC (tarjeta intermedia FPGA) y encabezados de expansión compatibles con Digilent Pmod (Peripheral Module interface), así como muchas características comunes utilizadas en el diseño de SoCs. ZedBoard habilita la capacidad de computación integrada mediante el uso de memoria DDR3, memoria Flash, Gigabit Ethernet, E/S de propósito general y tecnologías UART.

Las características de capacidad de expansión de esta plataforma de evaluación y desarrollo lo hacen ideal para la creación y desarrollo rápido de prototipos.

8.1.2.2. Características

Las características proporcionadas por la ZedBoard consisten en:

- System on Chip: Xilinx XC7Z020-1CLG484CES Zynq-7000 SoC:
  - Configuración principal = QSPI Flash
  - Opciones de configuración auxiliar:
    - JTAG en cascada
    - Tarjeta SD
- Memoria:
  - 512 MB DDR3 (128M x 32)
  - 256 Mb QSPI Flash
- Interfaces:
  - Programación USB-JTAG utilizando un circuito equivalente a SMT1 de Digilent
  - Accede a PL JTAG.
  - Pines PS JTAG conectados a través de PS Pmod
  - 10/100 / 1G Ethernet
  - USB OTG 2.0
  - Tarjeta SD
  - USB 2.0 FS puente USB-UART
  - Cinco cabeceras compatibles con Digilent Pmod ™ (2x6) (1 PS, 4 PL)
  - Un LPC FMC
  - Un encabezado de AMS
  - Dos botones de reinicio (1 PS, 1 PL)
  - Siete Botones Pulsadores (2 PS, 5 PL)
  - Ocho interruptores dip / switches (PL)
  - Nueve LEDs de usuario (1 PS, 8 PL)
  - LED DONE (PL)
- Osciladores en la tarjeta:
  - 33.333 MHz (PS)
  - 100 MHz (PL)
- Pantalla / Audio:
  - Salida HDMI
  - VGA (Color de 12 bits)
  - Pantalla OLED 128x32
  - Entrada de línea de audio, salida de línea, auriculares, micrófono
- Alimentación:
  - Interruptor de encendido / apagado
  - 12V @ 5A regulador AC / DC
- Software:
  - ISE WebPACK Design Software
  - Licencia para ChipScope Pro (bloqueado en XC7Z020)

El diagrama de bloques se puede consultar en la siguiente Figura 15.

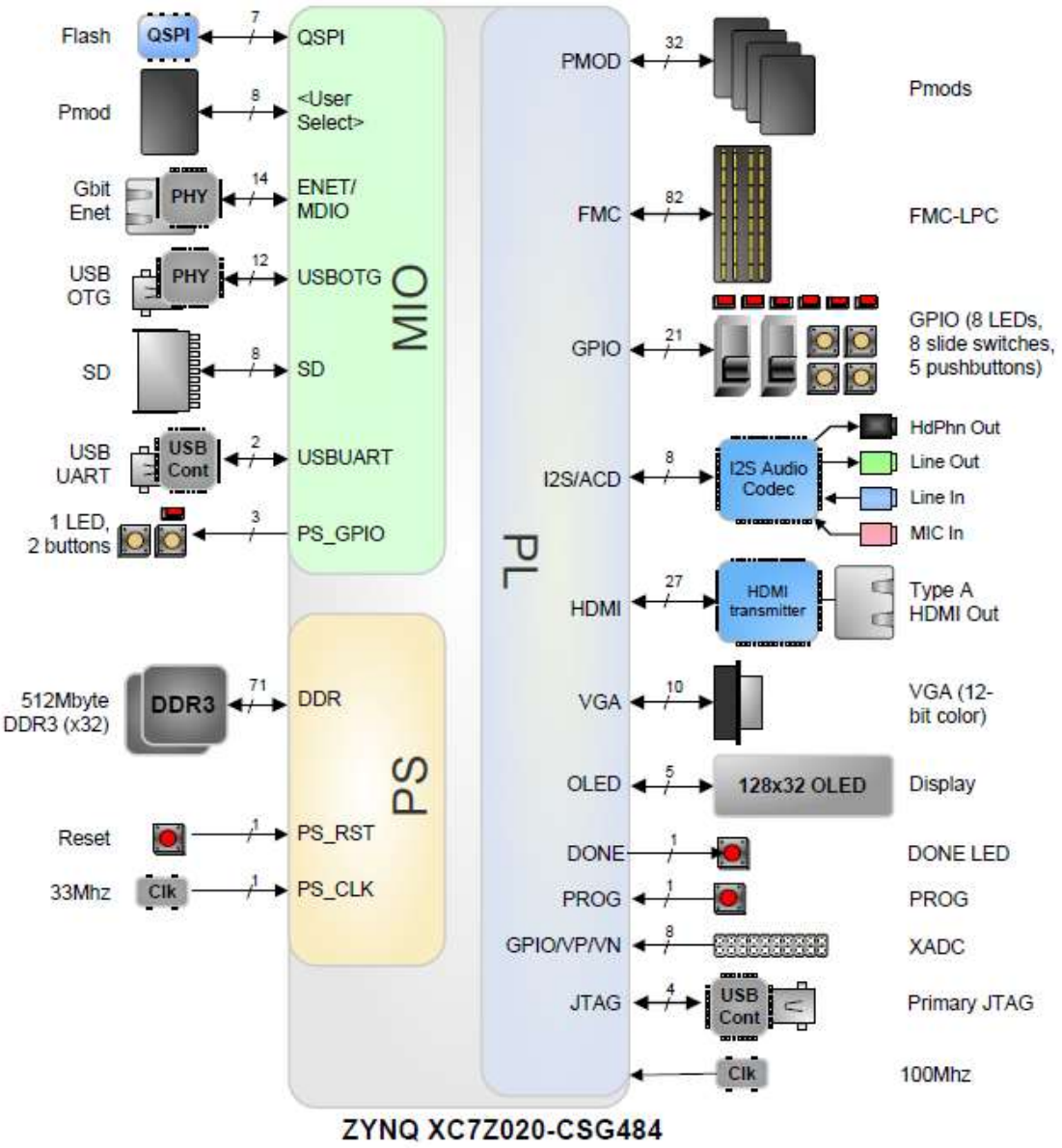


Figura 15: Diagrama de bloques de ZedBoard

Los pines están asignados a las E/S de la siguiente manera (Figura 16)

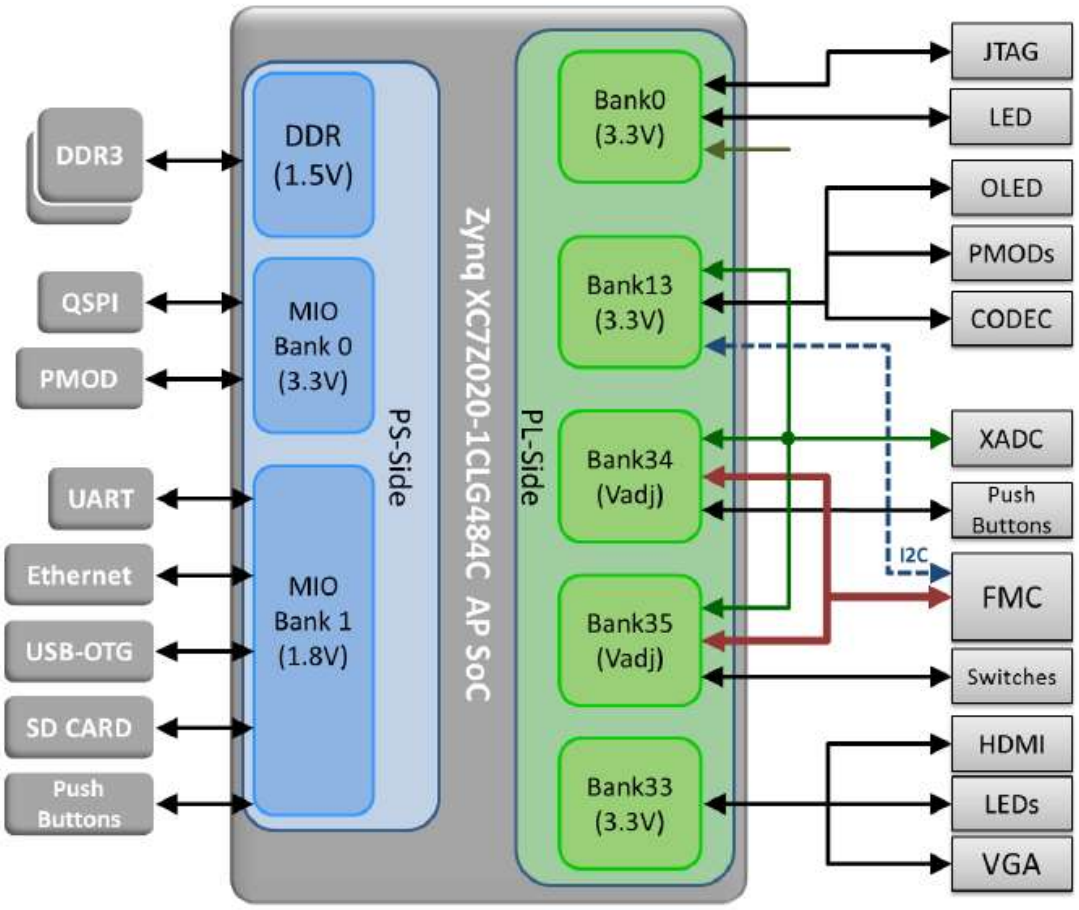


Figura 16: Asignación de pines a los bancos de la ZedBoard

8.1.2.3. Conexiones con las interfaces

En los siguientes apartados se detallarán las conexiones con las interfaces que se van a llevar a cabo durante este proyecto.

8.1.2.3.1. Conexión con tarjeta SD

El periférico Zynq PS SD/SDIO controla la comunicación con la tarjeta ZedBoard SD. La tarjeta SD se puede usar para el almacenamiento de memoria externa no volátil, así como para arrancar la Zynq-7000.

El PS periférico SD0 está conectado a través del Banco 1/501 MIO [40-47], incluyendo detección de tarjeta y protección contra escritura.

La tarjeta SD es una interfaz de 3.3V, pero está conectada a través del banco MIO 1/501 (1.8V). Por lo tanto, hace falta un cambiador de nivel TI TXS02612 que realice esta conversión. El TXS02612 es un expansor de puertos SDIO de 2 puertos con variador de nivel (Figura 17)

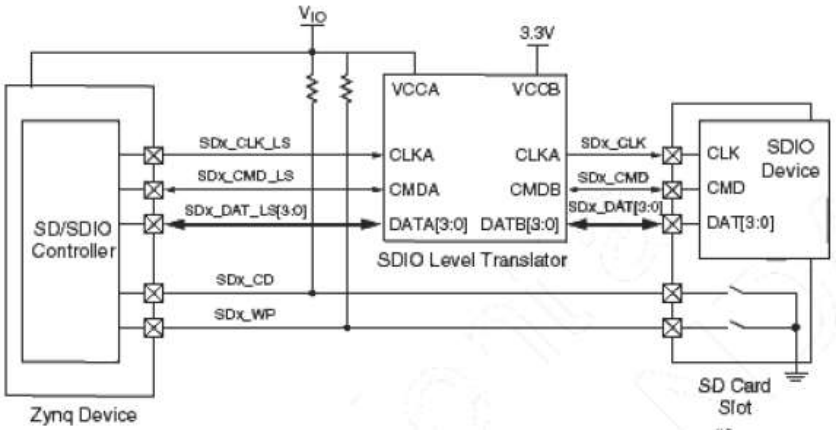


Figura 17: Conexiones del expansor de puertos SDIO

La tarjeta SD de ZedBoard se conecta a través de un conector de tarjeta SD estándar de 9 patillas (se conecta al conector J12). Se recomienda una tarjeta de Clase 4 o superior. Para poder usar la tarjeta SD como arranque, es necesario que el pin JP6 este cortocircuitado.

8.1.2.3.2. Conexión con USB

ZedBoard implementa una de las dos interfaces PS USB OTG (On the Go) disponibles. Se requiere un PHY (physical layer o capa física) externo con una interfaz ULPI de 8 bits (Figura 18) y se utiliza un chip de transceptor USB autónomo TI TUSB1210 como PHY.

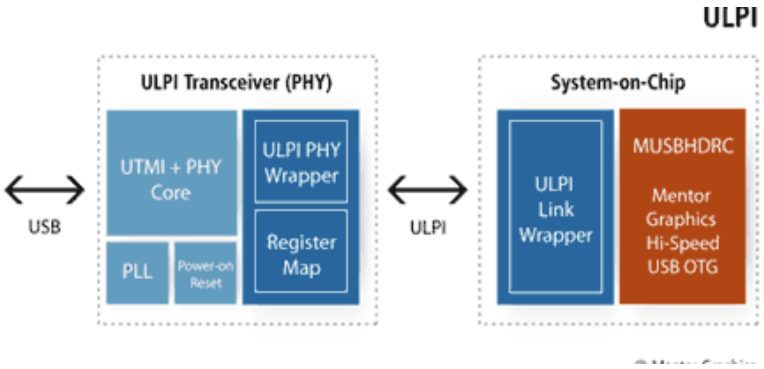


Figura 18: Conexión USB con FPGA

El PHY cuenta con un completo software físico USB que soporta velocidades de hasta 480Mbs. La tensión de alimentación para este dispositivo (VCCio) para este dispositivo es 1.8V y no se puede conectar a través de los cambiadores de nivel. El PHY está conectado al banco MIO 1/501, que funciona a 1.8V. Además, el chip USB debe sincronizar la interfaz ULPI que requiere un oscilador. Se utiliza un oscilador Fox XPRESSO (767-26-31) en ZedBoard.

La interfaz USB externa se conecta a través de un TE 1981584-1 (Figura 19).

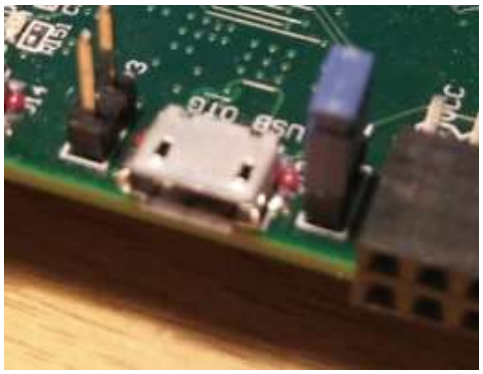


Figura 19:Interfaz de conexión externa USB TE 1981584-1

El periférico USB0 se utiliza en el PS, conectado a través de MIO [28-39] en MIO Bank 1/501.

8.1.2.3.3. Conexión con VGA

ZedBoard también permite una salida de video en color de 12 bits a través de un conector VGA. Se utiliza el conector TE 4-1734682-2 (Figura 20)

Figura 20: Conector TE 4-1734682-2

Cada color se crea a partir de la resistencia de cuatro pines PL que deben ser definidos en las restricciones del proyecto.

Se utiliza un conector DB 15 (Figura 21) en el que cada pin tiene asociada una señal (Tabla 2)

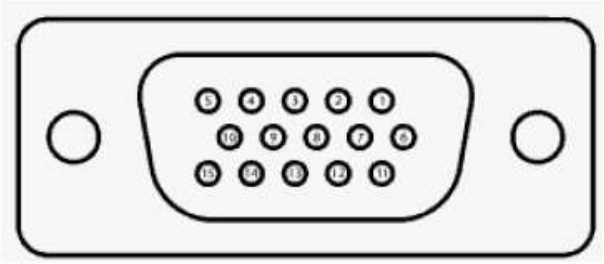


Figura 21: Conector para VGA DB 15

Pin VGA	Señal	Descripción	Zynq Pin
1	RED	Señal roja para video	V20, U20, V19, V18
2	GREEN	Señal verde para video	AB22, AA22, AB21, AA21
3	BLUE	Señal blue para video	Y21, Y20, AB20, AB19
4	ID2/RES	Anterior monitor ID bit 2	NC
5	GND	Tierra (HSync)	NC
6	RED_RTN	Señal de retorno de rojo	NC
7	GREEN_RTN	Señal de retorno de verde	NC
8	BLUE_RTN	Señal de retorno de azul	NC
9	KEY/PWR	Anterior key	NC
10	GND	Tierra (VSync)	NC
11	ID0/RES	Anterior monitor ID bit 0	NC
12	ID1/SDA	Anterior monitor ID bit 1	NC
13	HSync	Sincronismo horizontal	AA19
14	VSync	Sincronismo vertical	Y19
15	ID3/SCL	Anterior monitor ID bit 3	NC

Tabla 2: Pines VGA

8.1.2.3.4. Conexión con XADC

El XADC proporciona conectividad analógica para diseños de referencia analógicos, incluidas tarjetas secundarias AMS (Analog Mixed Signal) como la Tarjeta de evaluación AMS de Xilinx.

Tanto las E/S analógicas como las digitales pueden admitirse fácilmente. Esto permite que la cabecera analógica se conecte fácilmente a la tarjeta FMC mediante un

cable plano corto como se muestra (Figura 22). El encabezado analógico también se puede utilizar "independiente" para admitir la conexión de señales analógicas externas.

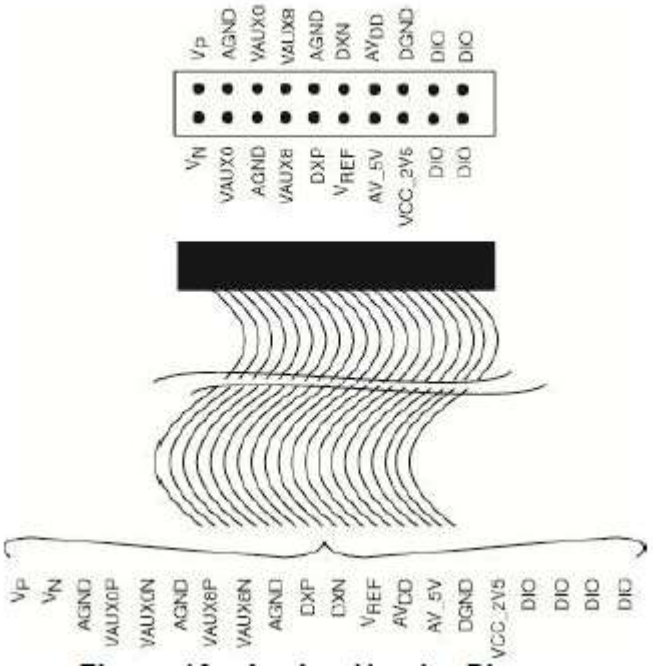


Figura 22:Entradas analógicas del XADC

El pin-out se ha elegido para proporcionar pares analógicos diferenciales estrechamente acoplados en el cable plano y también para proporcionar aislamiento entre canales.

El encabezado ZedBoard AMS es parecido a otras placas Xilinx como la KC705 y la ZC702. Cualquier tarjeta de complemento AMS construida para estas tarjetas también debería ser compatible con ZedBoard.

Se han los siguientes filtros de suavizado (antialiasing) para las entradas de XADC (Figura 23):

- VP / VN
- VAUX0P / VAUX0N
- VAUX8P / VAUX8N

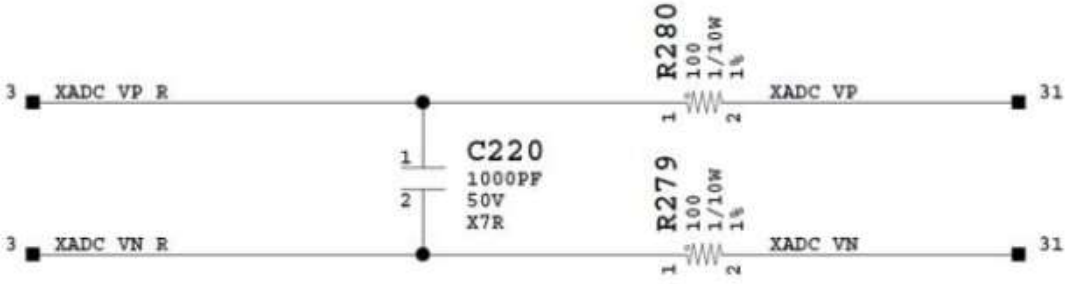


Figura 23:Filtros anti-aliasing de las entradas XADC

La siguiente Tabla 3 muestra el pin-out del XADC y sus requerimientos de tensión y temperatura.

Nombre	Descripción	Requerimientos	XADC Pines	Zynq Pines
VP/VN	Se requieren dos pines dedicados en el paquete de la serie 7. Este es el canal de entrada analógica dedicado para los ADC.	Máximo 1V de pico a pico de entrada	1 2	XADC-VN-R: M12 XADC-VP-R : L11
VAUX0P/ VAUX0N	Se requieren dos pines. Canal de entrada analógica auxiliar 0. También debe admitir el uso como entradas de IO mediante la desconexión del filtro antialiasing	Máximo 1V de pico a pico de entrada	6 3	XADC-AD0N-R: E16 XADC-AD0P-R : F16
VAUX8P/ VAUX8N	Se requieren dos pines. Canal de entrada analógica auxiliar 8. También debe admitir el uso como entradas de IO mediante la desconexión del filtro antialiasing	Máximo 1V de pico a pico de entrada	7 8	XADC-AD8N-R: D17 XADC-AD8P-R : D16
DXP/ DXN	Dos pines requeridos. Acceso a los diodos térmicos		12 9	XADC-DXN: N12 XADC-DXP : N11
AGND	Se requieren tres pines. Referencia analógica a tierra GNDADC. Aislamiento de canal analógico.		4 5 10	
VCCADC	Un pin. Esta es la fuente analógica de 1.8V para XADC.	1.8V ±5% @150mA máx.	14	
VREF	Un pin Esta es la referencia de 1.25V de la tarjeta	1.25V ±0.2% 50ppm/ºC @5mA máx.	11	
AV_5V	Suministro filtrado de 5V desde tarjeta	5V ±5% @150mA máx.	13	

GPIO	General Purpose I/O	Voltaje establecido por Vadj	G0: 18 G1: 17 G2: 20 G3: 19	H15 R15 K15 J15
Vadj	Voltaje ajustable, fijado por J18	1.8V, 2.5V, 3.3V		

Tabla 3: pin-out del XADC

8.1.2.4. Modos de configuración

Los dispositivos Zynq-7000 utilizan un proceso de inicio de múltiples etapas que admite tanto el inicio no seguro como el seguro. El PS es el maestro del proceso de inicio y configuración. La siguiente tabla muestra los modos de configuración de Zynq. Al reiniciar, se leen los pines de modo del dispositivo para determinar el dispositivo de arranque primario que se utilizará. Los posibles modos de arranque son:

- Quad-SPI
- Tarjeta SD
- JTAG.

De forma predeterminada, la tarjeta ZedBoard utiliza el modo de configuración de la tarjeta SD. Los pines del modo de inicio son MIO [8: 2] (Figura 24) y se utilizan de la siguiente manera (Tabla 4):

Multiplexed Input Output (MIO)	**Pin ZedBoard**	**Boot Mode**	**Descripción**
MIO[2]	JP7	Boot_Mode [3]	Establece el modo JTAG
MIO[5:3]	JP10,JP9,JP8	Boot_Mode [2:0]	Selecciona el modo de inicio (Tabla 5)
MIO[6]	JP11	Boot_Mode [4]	Habilita el PLL interno
MIO [8:7]		Vmode [1: 0]	Se utilizan para configurar los voltajes del banco de E/S. Estos están fijos en ZedBoard y no son configurables

Tabla 4: Modos de arranque de la ZedBoard

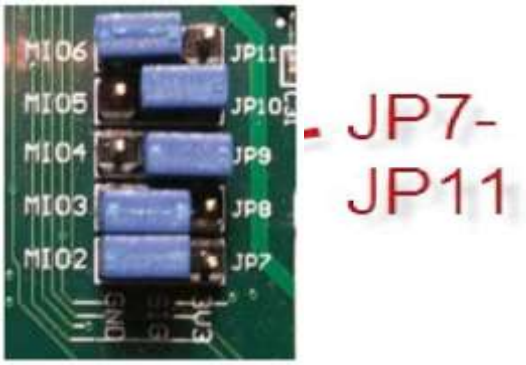


Figura 24: Entradas/salidas multiplexadas (MIOs) con sus respectivos pines

Xilinx TRM→	MIO[6] Boot Mode[4]	MIO[5] Boot Mode[2]	MIO[4] Boot Mode[1]	MIO[3] Boot Mode[0]	MIO[2] Boot Mode[3]
JTAG Mode					
Cascaded JTAG					0
Independent JTAG					1
Boot Devices					
JTAG		0	0	0	
Quad-SPI		1	0	0	
SD Card		1	1	0	
PLL Mode					
PLL Used	0				
PLL Bypassed	1				
Bank Voltages					
MIO Bank 500			3.3V		
MIO Bank 501			1.8V		

Tabla 5: Modos de configuración de la ZedBoard dependiendo de posición de los jumpers.

Estos jumpers tomaran un 1 lógico cuando estén conectados a 3,3 V y un 0 lógico cuando estén conectados a tierra GND (Figura 25)

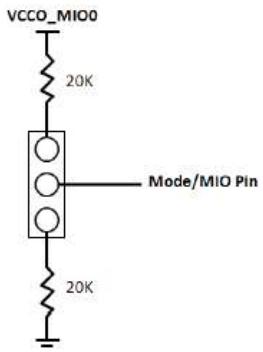


Figura 25: Conexión de los jumpers

El modo de arranque que es interesante para este proyecto es el de arranque mediante tarjeta SD. Para ello, según la Tabla 5. deben estar los jumpers configurados de la siguiente manera (Tabla 6):

JP11	0
JP10	1
JP9	1
JP8	0
JP7	0

Tabla 6: Modo de arranque por tarjeta SD

Además, si queremos que lea también un .bit, es necesario cortocircuitar el jumper JP6. Estos jumpers junto con el resto disponible se encuentran situados en la Figura 26.

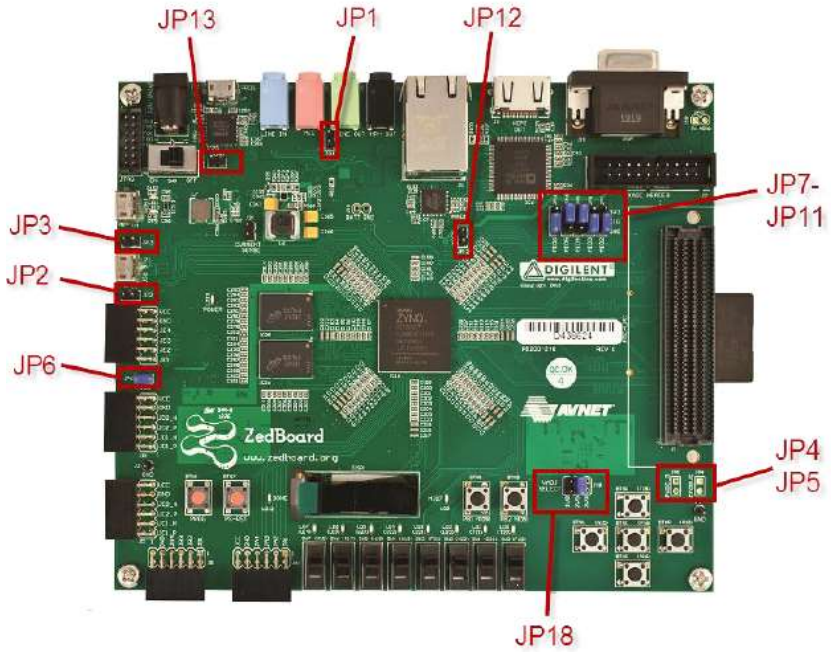


Figura 26: Situación de los jumpers en la ZedBoard

8.1.2.5. Herramienta de desarrollo hardware Vivado

Vivado es la herramienta principal de Xilinx para desarrollo en FPGA. Proporciona una interfaz gráfica de usuario desde la que es posible llevar a cabo todo el diseño y la síntesis de circuitos. Todas sus herramientas y opciones están escritas en formato Tcl (Tool Command Language), lo que hace posible usar tanto la interfaz gráfica como el intérprete de comandos.

Como la base de datos está accesible por Tcl, los cambios en las restricciones, configuraciones u opciones en las herramientas se aplican en tiempo real, sin necesidad

de forzar la reimplementación. Tiene muchas opciones que se pueden utilizar en cualquier etapa del diseño como definir restricciones de tiempo, explorar el catálogo de IP, etc. También ofrece realimentación del diseño en todo momento con la estimación de recursos usados, retraso de interconexión o consumo de energía.

Vivado, como se ha citado anteriormente, es posible descargarlo desde la página oficial de Xilinx https://www.xilinx.com/support/download.html.

Vivado Design Suite es una suite de software producida por Xilinx para la síntesis y el análisis de diseños HDL (hardware description lenguaje) y que tiene unas altas capacidades para un desarrollo de chips y síntesis de alto nivel.

La edición del sistema Vivado incluye un simulador lógico incorporado. Vivado también presenta síntesis de alto nivel, con una cadena de herramientas que convierte el código C en lógica programable. Vivado ha sido descrito como una herramienta de automatización de diseño electrónico de última generación con todas las últimas novedades en términos de modelo de datos, integración, algoritmos y rendimiento.

Vivado permite sintetizar sus diseños, realizar análisis de tiempo, examinar diagramas RTL (Register-transfer level), simular la reacción de un diseño a diferentes estímulos y configurar el dispositivo de destino con el programador. Vivado es un entorno de diseño para productos FPGA de Xilinx y está estrechamente relacionado con la arquitectura de dichos chips, y no se puede utilizar con productos FPGA de otros proveedores.

Vivado es un entorno de diseño integrado (IDE) que incluye herramientas de diseño de nivel de sistema electrónico para sintetizar y verificar la IP algorítmica basada en C; empaquetados basados en estándares de algoritmos y RTL IP para reutilización e integración de sistemas de todos los tipos de bloques de construcción de sistemas; y la verificación de bloques y sistemas.

8.1.2.5.1. Componentes de Vivado

El compilador de síntesis de alto nivel de Vivado permite que los programas C, C ++ y SystemC se dirijan directamente a los dispositivos Xilinx sin la necesidad de crear manualmente RTL.

El simulador de Vivado es un componente de la Suite de diseño de Vivado. Es un simulador de lenguaje compilado que admite scripts TCL de lenguaje mixto, IP cifrada y verificación mejorada.

El integrador de IP de Vivado permite integrar y configurar rápidamente la IP de la gran biblioteca de IP de Xilinx. El integrador también está optimizado para los diseños de MathWorks Simulink creados con el generador de sistemas de Xilinx y la síntesis de alto nivel de Vivado.

Vivado TLC Store es un sistema de lanzamiento de archivos por lotes (.bat) para el desarrollo de complementos a Vivado, y se puede utilizar para agregar y modificar las capacidades de Vivado. TCL (Tool Command Language) es el lenguaje de script en el que se basa Vivado. Todas las funciones que hay por debajo de Vivado pueden invocarse y controlarse mediante scripts TCL.

8.1.2.5.2. Soporte del dispositivo

A partir de 2018, Xilinx recomienda Vivado Design Suite para los nuevos diseños con Ultrascale, Ultrascale +, Virtex-7, Kintex -7, Artix -7 y Zynq-7000.

Vivado proporciona una integración y una implementación más rápidas para los sistemas programables en dispositivos con tecnología de interconexión de silicio apilado 3D, sistemas de procesamiento ARM, señal mixta analógica (AMS) y muchos núcleos de propiedad intelectual (IP).

Vivado está dirigido a los FPGA más grandes de Xilinx, y está reemplazando lentamente a Xilinx ISE como su cadena de herramientas principal. A partir de 2014, Vivado cubre las FPGA grandes y medianas de Xilinx, e ISE cubrió los FPGA más pequeños y medianos y todos los CPLD (Complex Programmable Logic Device).

8.1.3. XADC

Para el desarrollo de nuestra aplicación, es vital un sistema que pueda captar datos analógicos externos y convertirlos a números digitales que sea capaz de interpretar la FPGA. Por ello, se cuenta con un sistema de adquisición de datos XADC integrado en la Zynq-7000.

Es importante saber cómo funciona dicho conversor, ya que conocerlo nos dará opción a poder utilizar las múltiples configuraciones de funcionamiento que este tiene. A continuación, se hace un resumen del manual de funcionamiento adjuntado en el punto [17] del apartado 5.3. Bibliografía y enlaces web.

8.1.3.1. Visión general

El XADC incluye un ADC de doble bit y 1 mega muestra por segundo (MSPS) y sensores on chip, totalmente probados y especificados.

Los ADC proporcionan una interfaz analógica de alta precisión para fines generales para una amplia gama de aplicaciones. La Figura 27 muestra un diagrama de bloques del XADC.

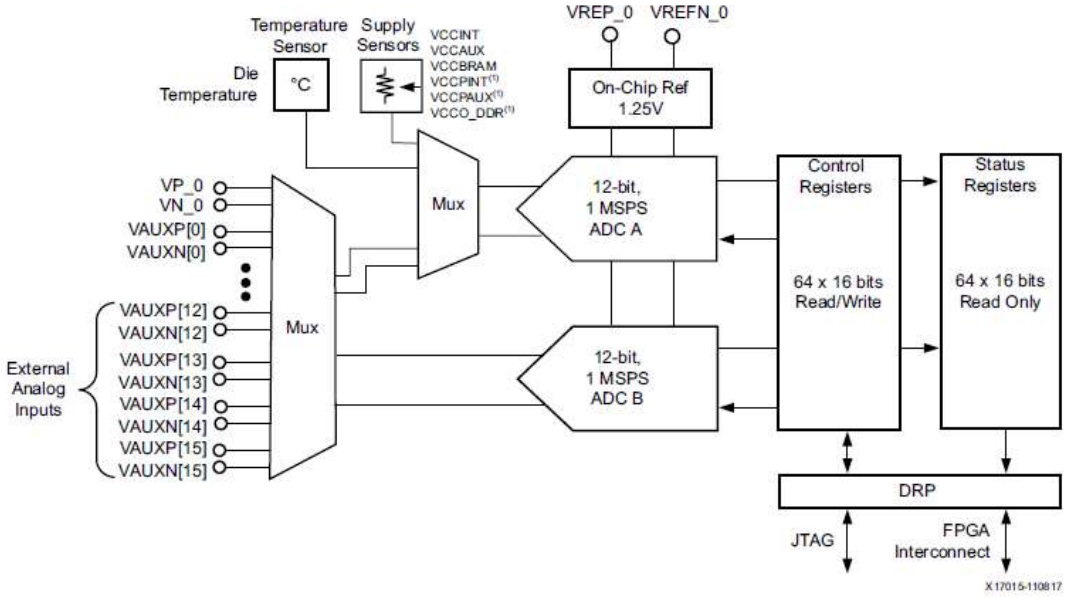


Figura 27: Diagrama de bloques del conversor ADC

Los ADCs duales admiten una gama amplia de modos de funcionamiento, por ejemplo, disparos externos y simultáneos, muestreo en ambos ADC, etc.

Además, se pueden tomar varios tipos de señales de entrada como, por ejemplo: unipolar, diferencial…

Los ADC pueden acceder a hasta 17 canales de entrada analógicos externos, siendo estos el canal dedicado VPVN y los canales auxiliares del 0 al 15.

Tal y como se ve en Figura 27, la salida que proporciona el ADC es de 12 bits, tanto para el conversor ADC A como para el ADC B.

También incluye sensores para leer la temperatura interna de la FPGA.

8.1.3.2. Requisitos de pin-out

Todos los pines dedicados de XADC están ubicados en el banco 0 y, por lo tanto, tienen el sufijo _0 cuando los instanciamos como IP Core.

La Figura 28 muestra los requisitos básicos de los pines para el XADC. Hay dos configuraciones recomendadas. A la izquierda, el XADC se alimenta desde VCCAUX (1.8V) y utiliza una fuente de referencia externa de 1.25V. La referencia externa entrega mejor rendimiento en términos de precisión y deriva térmica. Se utiliza una perla de ferrita para aislar la referencia de tierra para los circuitos analógicos y tierra del sistema. Un filtro de paso bajo, para el suministro de VCCAUX, mejorará de manera similar el rendimiento de ADC.

La otra opción es utilizar una referencia on chip para los ADC. Para habilitar la dicha fuente, el pin VREFP debe estar conectado a tierra como se muestra a la derecha de la Figura 28. Dicha referencia proporciona también un buen rendimiento.

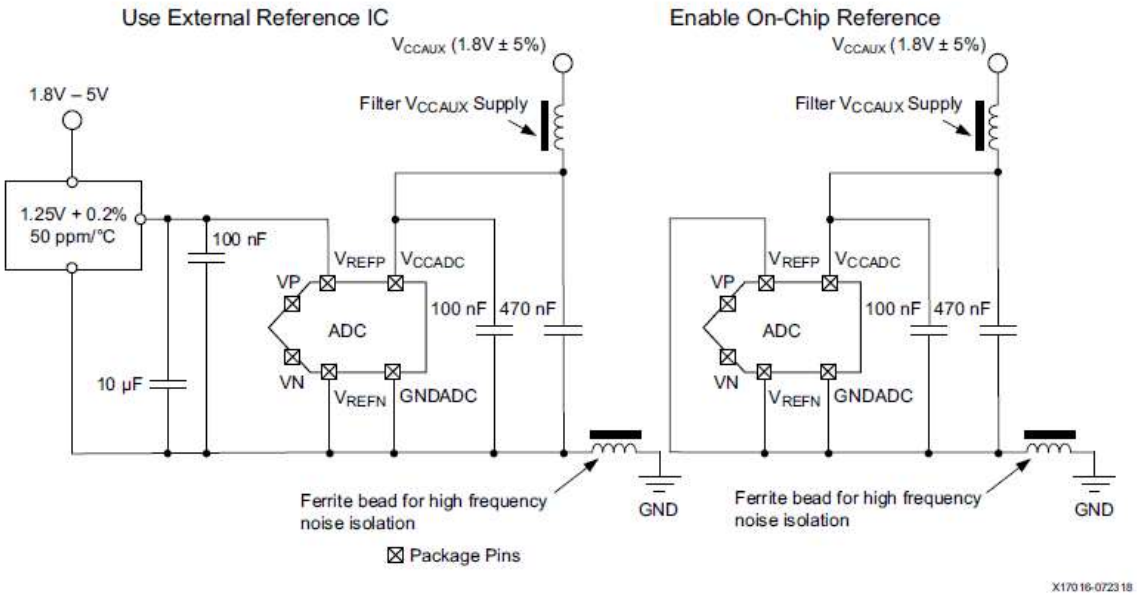


Figura 28: Requisitos de los pines del XADC

Aparte de un solo par de entradas analógicas dedicadas (VP / VN), las entradas analógicas externas utilizan E / S de doble propósito. Estas E / S digitales FPGA están nominadas individualmente como entradas analógicas. Cuando el XADC es instanciado en un diseño. Este documento se refiere a estas entradas analógicas como entradas analógicas auxiliares.

Además del canal de entradas analógicas dedicado VP / VN, se cuenta con 16 entradas auxiliares que pueden ser habilitadas.

Todos los canales de entrada analógicos son diferenciales y requieren dos paquetes de bolas. Las entradas analógicas auxiliares se asignan de manera uniforme en los bancos 15 y 35 y tienen dos paquetes fijos de asignación de bola que no se pueden mover.

Las E / S con capacidad analógica tienen el sufijo ADxP o ADxN en el nombre de E / S en los archivos del paquete. Por ejemplo, el canal de entrada analógica auxiliar 8 tiene nombres de bola de paquete asociados que terminan con AD8P y AD8N.

8.1.3.3. Instanciación del XADC

No es necesario crear una instancia de XADC en un diseño para poder ser capaces de monitorizar lo que muestra el chip. Sin embargo, si el XADC no es instanciado en el diseño, la única forma de acceder a esta información es a través del puerto de acceso de prueba JTAG. Para permitir acceso a los registros de estado (resultados de medición) desde la propia FPGA, el XADC debe ser instanciado

8.1.3.3.1. Puertos del XADC

En la siguiente Figura 29. se muestran todos los puertos accesibles del XADC.

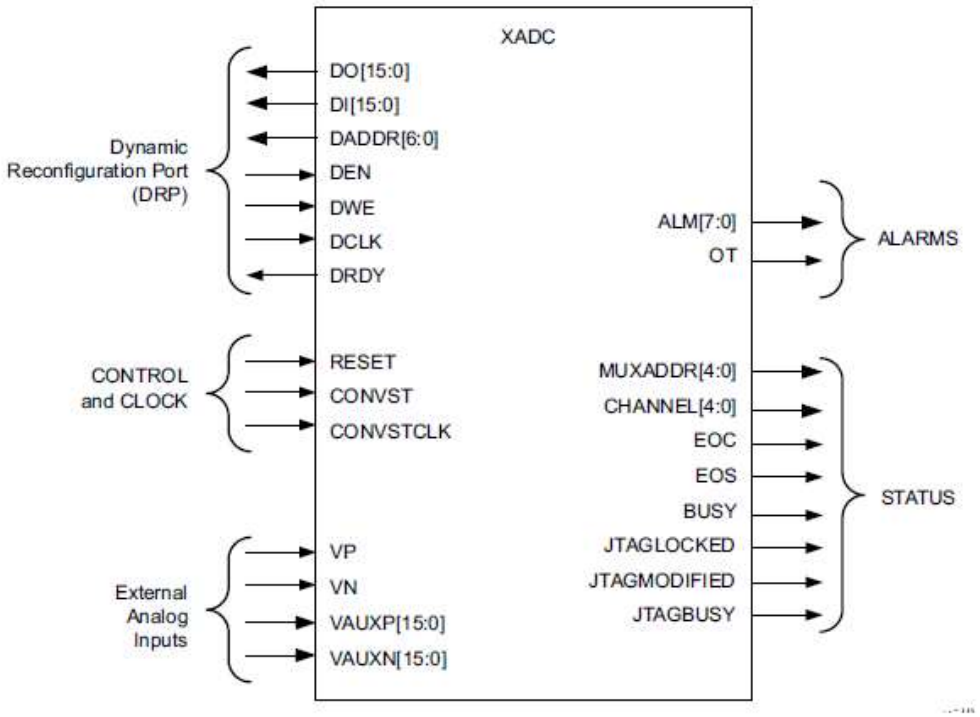


Figura 29: Puertos accesibles del XADC

Se adjunta la Tabla 7 que define la función de cada uno de los puertos, tanto de entrada como de salida.

Puerto	E / S	Descripción
DI [15:0]	E	Bus de datos de entrada para el DRP
DO [15:0]	S	Bus de datos de salida para el DRP
DADDR [6:0]	E	Dirección para el DRP
DEN	E	Habilitar señal DRP
DWE	E	Escribir habilitación DRP

DCLK	E	Entrada de clock para DRP
DRDY	S	Señal de datos preparados para el DRP
RESET	E	Reset asíncrono
CONVST	E	Entrada de inicio de conversión de datos utilizado para el modo Disparo por evento
CONVSTCLK	E	Controla el instante de inicio de muestreo, pero a diferencia con la anterior, está conectada a un reloj
VP,VN	E	Canal de entradas analógicas dedicado
VAUXP,VAUXN[15:0]	E	16 canales de entradas analógicas auxiliares
ALM[7:0]	S	Alarmas para los diferentes sensores
OT	S	Alarma por exceso de temperatura
MUXADDR [4:0]	S	Usadas en el modo multiplexor externo e indican la dirección del próximo canal en la secuencia en la que deben ser convertidos.
CHANNEL [4:0]	S	Saca por esta salida el canal que está convirtiendo en el momento actual.
EOC	S	Fin de conversión
EOS	S	Fin de secuencia
BUSY	S	Ocupado
JTAGLOCKED	S	Indica que el DRP ha sido requerido para usarlo por el JTAG
JTAGMODIFIED	S	Indica que el JTAG ha escrito en el DRP
JTAGBUSY	S	Indica que hay una transacción JTAG-DRP en progreso

Tabla 7: Función de los puertos

No todas las entradas o salidas son útiles para el proyecto. Es necesario estudiar en cada caso que entrada es interesante, pudiendo ocultar y no mostrar las que no lo son. Esto es posible cuando accedemos al XADC a través del DRP instanciado en Vivado mediante un IP Core.

Además de poder acceder al mediante el DRP, se puede acceden también mediante un periférico AXI que utiliza el micro ARM y a través del JTAG.

8.1.3.3.2. Registros del XADC

Los registros del ADC se dividen en dos bloques principales: los registros de estado (00h a 3Fh) de solo lectura en los que encontramos el valor de los datos convertidos y los registros de control (40h a 5Fh) a través de los cuales se puede determinar el funcionamiento del ADC. Se muestra dicho esquema de registros en la Figura 30.

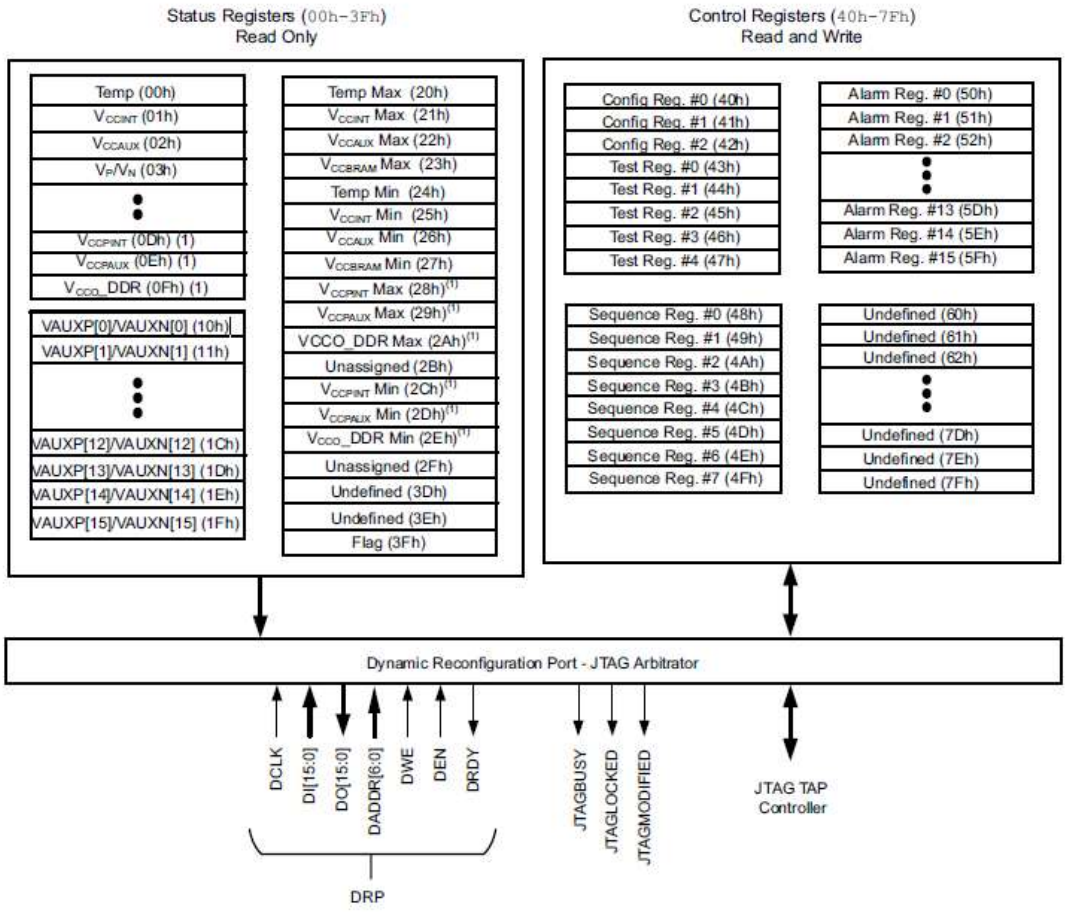


Figura 30: Esquema de los registros del XADC

De los registros de estado nos interesan los valores de las conversiones de los canales que se utilizaran posteriormente como son el canal dedicado VPVN cuyo valor convertido se encuentra en el registro 03h y el valor de los canales auxiliares VAUX [0:15] cuyos resultados se encuentran en los registros del 10h al 1Fh.

Los siguientes registros de control de la Tabla 8 configuran el funcionamiento del XADC según los parámetros que se necesiten y sean establecidos. Se trata de 32 registros de 16 bits cada uno que pueden ser escritos o leído a través de los puertos DRP o JTAG. Estos atributos se llaman INIT_XX, siendo _XX el valor de la dirección hexadecimal del registro.

Atributo	Nombre	Dirección del registro
INIT_40	Registro de configuración	0x40
INIT_41	Registro de configuración	0x41
INIT_42	Registro de configuración	0x42
INIT_43 a INIT_47	Registro de test	0x43 a 0x47
INIT_48 a INIT_4F	Registro de secuencia	0x48 a 0x4F
INIT_50 a INIT_5F	Registro de límites de alarmas	0x50 a 0x5F

Tabla 8:Registros y direcciones del XADC

Los registros de configuración (40h, 41h y 42h) se encargan de seleccionar los canales de muestreo, la media de cuantas muestras se debe sacar, las alarmas, las secuencias (continua, única, etc.) Para modificar cada uno de los parámetros que se deseen ajustar, se deben añadir 0 o 1 en los bits adecuados de dichos registros que se muestran en la Figura 31.

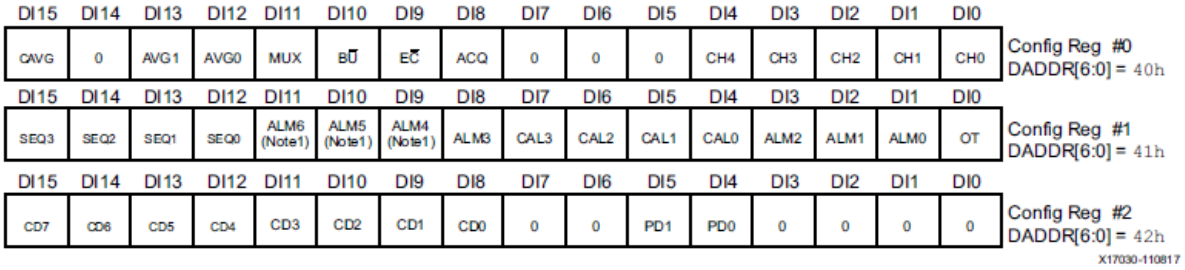


Figura 31: Registros de configuración del XADC

A continuación, se muestra un ejemplo de cómo se inicializarían los registros de configuración junto con su significado.

```
-- INIT_40 - INIT_42: XADC configuration registers
INIT_40 => X"1013", -- CAVG averaging of 16 samples in calib coeff, amount of avg en
INIT_41 => X"2020", -- Single chanel Mode, Disable  ALMs, Disable calibration
INIT_42 => X"6400", -- ADCCLK = DCLK/100 = 100MHz/100 = 1 MHz
```

Los registros de test (43h a 47h) están destinados únicamente a fines de prueba de fábrica y tienen por defecto un valor de ‘0000’ que no se debe modificar

Si en los registros de configuración se ha hecho uso del modo automático, existen los registros de secuencia (48h a 4Fh). Estos registros se dividen en 4 partes:

- **Selector de canal (48h y 49h):** se encargan de seleccionar los canales que el usuario quiere que sea convertido. Se utiliza el D11 del registro 48h (Figura 32) para activar el canal dedicado VPVN y los bits D0 a D15 para las entradas auxiliares (Figura 33)

Sequencer On-Chip Channel Selection (48h)

Sequence Number 7 Series/Zynq-7000	Bit	ADC Channel	Description
5/8	11	3	V_P, V_N – Dedicated analog inputs

Figura 32: Registro para activar el canal dedicado VPVN

Sequence Number 7 Series/Zynq-7000	Bit	ADC Channel	Description
9/12	0	16	VAUXP[0], VAUXN[0] – Auxiliary channel 0
10/13	1	17	VAUXP[1], VAUXN[1] – Auxiliary channel 1
11/14	2	18	VAUXP[2], VAUXN[2] – Auxiliary channel 2
12/15	3	19	VAUXP[3], VAUXN[3] – Auxiliary channel 3
13/16	4	20	VAUXP[4], VAUXN[4] – Auxiliary channel 4

Figura 33: Registro para activar los canales auxiliares

- **Promediado (4Ah y 4Bh):** activan o desactivan el promediado de los canales que se hayan establecido en el registro anterior (poner su misma configuración).
- **Modo de entrada analógica (4Ch y 4Dh):** Estos registros se utilizan para configurar un canal ADC como unipolar o bipolar en la secuencia automática Estos registros también tienen la misma asignación de bits como los registros de secuencia de canales (48h y 49h). Sin embargo, solo canales de entrada analógicos externos, como los canales de entrada dedicados (VP y VN) y las entradas auxiliares analógicas (VAUXP [15: 0] y VAUXN [15: 0]) se pueden configurar de este modo. Colocar un 1 en el bit dedicado significa que lo configuramos en modo bipolar, mientras que un 0 significa modo unipolar.
- **Tiempo de establecimiento:** por defecto, el tiempo de establecimiento para una secuencia continua es de 4 ciclos de ADCCLK pudiéndose ampliar hasta 10 ciclos escribiendo un 1 en la posición correspondiente al canal dedicado. El tiempo de establecimiento es el tiempo que tarda en poder volver a captar otra muestra después de la última conversión. La asignación de bits es igual que en los tres casos anteriores.

En los registros de alarma (50h a 5Fh) se establecen los limites tanto superior como inferior de las alarmas a los cuales deberían saltar. Los registros son los que se muestran en la siguiente Figura 34.

Control Register	Description	Alarm
50h	Temperature upper	ALM[0]
51h	V_{CCINT} upper	ALM[1]
52h	V_{CCAUX} upper	ALM[2]
53h	OT alarm limit(1)	OT
54h	Temperature lower	ALM[0]
55h	V_{CCINT} lower	ALM[1]
56h	V_{CCAUX} lower	ALM[2]
57h	OT alarm reset(1)	OT
58h	V_{CCBRAM} upper	ALM[3]
59h	V_{CCPINT} upper(2)	ALM[4]
5Ah	V_{CCPAUX} upper(2)	ALM[5]
5Bh	V_{CCO_DDR} upper(2)	ALM[6]
5Ch	V_{CCBRAM} lower	ALM[3]
5Dh	V_{CCPINT} lower(2)	ALM[4]
5Eh	V_{CCPAUX} lower(2)	ALM[5]
5Fh	V_{CCO_DDR} lower(2)	ALM[6]

Figura 34:Registro de alarmas del XADC

8.1.3.4. Timing del XADC

La temporización XADC se sincroniza con el reloj DRP (DCLK). El ADCCLK es generado dividiendo DCLK por la selección del usuario en el registro de configuración 2 (registro 42h). ADCCLK es un reloj interno utilizado por los ADC y no está disponible externamente.

Existen dos modos de operación del XADC:

- **Modo continuo**: en el cual el ADC automáticamente empieza una nueva conversión cuando acaba el ciclo de conversión actual. El proceso de conversión de analógico a digital está compuesto por dos fases: la fase de adquisición y la fase de conversión. La secuencia de muestreo de una tensión se ve reflejada en la Figura 35.

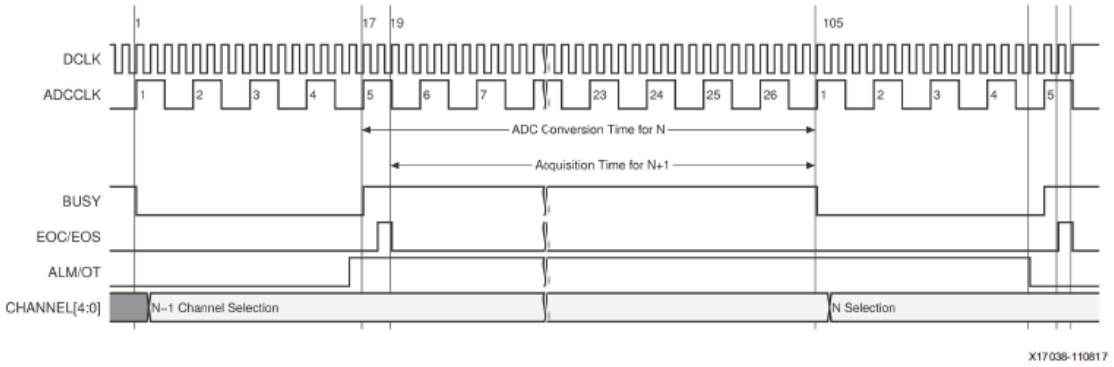


Figura 35:Muestreo Continuo

- **Modo Disparo por Evento**: en este modo de funcionamiento, el instante de muestreo y el proceso de conversión posterior se inician mediante una señal de activación llamada Convertir Inicio (CONVST). El modo de muestreo controlado por evento se usa cuando el control preciso sobre el instante de muestreo es necesario. El diagrama temporal de dicho modo se representa en la Figura 36.

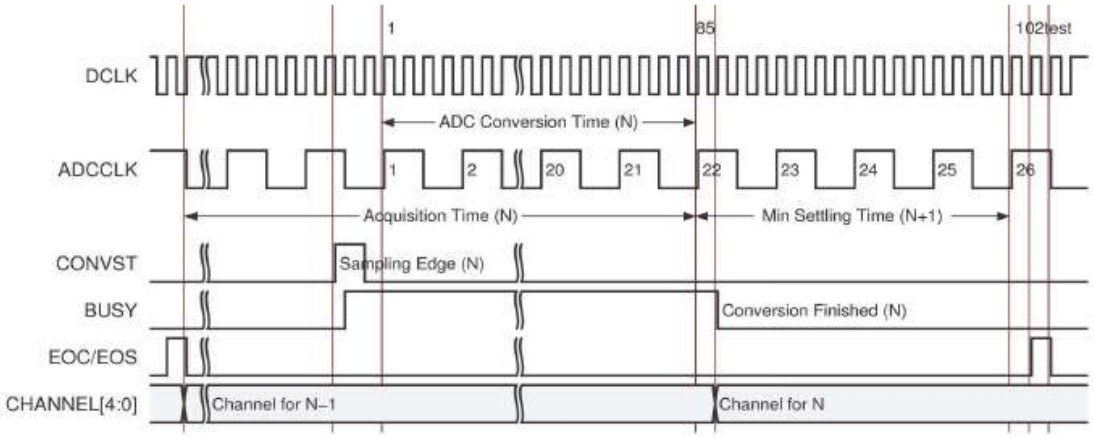


Figura 36:Muestreo por Eventos

Analizando las Figuras 35 y 36, se ve que la conversión no finaliza cuando se ha adquirido la muestra, es decir, ha pasado el tiempo de adquisidor, sino que finaliza un tiempo más tarde. Se sabe que ha finalizado la conversión cuando la señal EOC se pone a nivel alto durante un periodo de tiempo. Hasta que se produce dicho flanco ascendente de EOC (End of Conversion) han pasado 2 tiempos diferentes para la misma muestra. Esto es debido a que el muestreo se compone de dos fases: fase de adquisición (en la cual se carga un condensador del canal determinado con el voltaje que se encuentra en la entrada) y fase de conversión (en la cual se convierte el valor obtenido en el condensador a los 16 bits de salida. Este periodo ocupa 22 ADCCLK y permite que se produzca la fase de adquisición del canal siguiente).

En la configuración, la entrada DCLK requiere 20 DCLK para volver a sincronizarse con el reloj externo. La señal JTAGLOCKED se mantiene a nivel alto durante este periodo, por lo que no es posible acceder al DRP hasta que se haya producido la resincronización. Si se está utilizando el modo de secuenciador automático, por la salida CHANNEL, se podrá comprobar que canal es el que se está convirtiendo en cada momento.

8.1.3.5. Modos de operación

Los diferentes modos de operación se configuran desde los bits SEQ3 a SEQ0 del registro 41h de los registros de configuración. Se puede ver como activar cada modo en la siguiente 37.

SEQ3	SEQ2	SEQ1	SEQ0	Function
0	0	0	0	Default mode
0	0	0	1	Single pass sequence
0	0	1	0	Continuous sequence mode
0	0	1	1	Single channel mode (sequencer off)
0	1	X	X	Simultaneous sampling mode
1	0	X	X	Independent ADC mode
1	1	X	X	Default mode

Figura 37:Diferentes modos de operación del XADC.

- **Modo canal único**: se habilita desde los registros de configuración (41h) y selecciona el único canal a convertir en el registro 40h. En aplicaciones donde muchos canales deben ser monitorizados, puede haber una sobrecarga del microprocesador, por lo que se recurre al modo secuenciador automático.
    - **Modo secuenciador automático de canal:** se configura en los registros 41h de configuración y en los de secuencia (48h a 4Fh) y se utiliza para convertir de forma automática varios canales. Los modos de secuencia que se pueden utilizar ajustando la configuración adecuada son los siguientes:
    - **Modo por defecto**: se habilita al colocar un 0 en los bits de secuencia. En este modo el XADC supervisa automáticamente los sensores on chip y almacena los resultados los registros correspondientes (ver Figura 30). Ambos ADCs están calibrados en este modo y se aplica un promedio de 16 muestras a todos los sensores. El XADC también funciona en modo predeterminado después del encendido inicial y durante la configuración de la FPGA. La siguiente Figura 38 muestra el orden de muestreo de los canales.

Order	Channel	Address	Description
1	Calibration	08h	Calibration of ADC A and ADC B
2(1)	V_{CCPINT}	0Dh	V_{CCPINT} supply sensor
3(1)	V_{CCPAUX}	0Eh	V_{CCPAUX} supply sensor
4(1)	V_{CCO_DDR}	0Fh	V_{CCO_DDR} supply sensor
2/5(1)	Temp	00h	Temperature sensor
3/6(1)	V_{CCINT}	01h	V_{CCINT} supply sensor
4/7(1)	V_{CCAUX}	02h	V_{CCAUX} supply sensor
5/8(1)	V_{CCBRAM}	06h	V_{CCBRAM} supply sensor

Figura 38: Orden de muestreo de los canales en el modo por defecto

- **Modo Paso Único**: realiza la secuencia de monitorización de los canales especificados en los registros de configuración una única vez. Tras finalizar la conversión de los N canales seleccionados, pasa al modo de canal único. Para volver a activar el modo de paso único es necesario escribir de forma manual un 0001 en los bits SEQ3 a SEQ0 del registro 41h, tal y como se muestra en la Figura 37.
- **Modo Secuencia Continua**: es igual que el anterior, pero con la diferencia de que, tras finalizar la conversión de las muestras, no vuelve al modo Single Channel, sino que vuelve a muestrear de nuevo los mismos canales.
- **Modo Muestreo Simultaneo**: el secuenciador muestrea automáticamente a través de ocho pares de canales de entrada analógicos auxiliares. Estos canales de muestreo se ven reflejados en la Figura 39. Esto es útil en aplicaciones donde es necesario preservar la relación de fase entre dos señales. Los canales analógicos auxiliares 0 a 7 se asignan a ADC A y se nombran como canales A. Los canales analógicos auxiliares 8 a 15 se asignan a ADC B y se nombran como canales B. Un canal A y un canal B siempre se muestrean y se convierten al mismo tiempo en modo de muestreo simultáneo.

Sequence Number	Bit	ADC Channel	Description
1	0	16, 24	Auxiliary channels 0 and 8
2	1	17, 25	Auxiliary channels 1 and 9
3	2	18, 26	Auxiliary channels 2 and 10
4	3	19, 27	Auxiliary channels 3 and 11
5	4	20, 28	Auxiliary channels 4 and 12
6	5	21, 29	Auxiliary channels 5 and 13(1)
7	6	22, 30	Auxiliary channels 6 and 14(1)
8	7	23, 31	Auxiliary channels 7 and 15(1)
x	8	x	Undefined
x	9	x	Undefined
x	10	x	Undefined
x	11	x	Undefined
x	12	x	Undefined
x	13	x	Undefined
x	14	x	Undefined
x	15	x	Undefined

Figura 39: Canales del modo Muestreo Simultaneo

- **Modo ADC Independiente**: En el modo ADC independiente, ADC A se usa para implementar un "modo de monitoreo" fijo, similar al modo predeterminado (excepto que las funciones de alarma están habilitadas y deben ser configuradas correctamente ajustándole los umbrales de alarma en los registros de alarma 50h a 5Fh).

  El ADC B está disponible para ser utilizado solo con los canales de entrada analógicos externos. Este modo libera el segundo ADC para su uso en una aplicación de cliente donde solo un ADC es requerido mientras se mantiene el monitoreo de la FPGA para proporcionar seguridad contra manipulaciones.

  Solo el canal dedicado y los canales de entrada analógica auxiliar pueden asignarse al ADC B. Los canales internos (sensores) se asignan automáticamente al ADC A, que supervisa automáticamente estos canales y genera alarmas basadas en los umbrales establecidos por el usuario.

  El XADC se puede devolver al modo ADC independiente si EOS (End of Sequence) se pone a nivel alto al menos una vez.

- **Modo multiplexador externo:** este modo se utiliza para en lugar de consumir las 32 E / S requeridas para implementar los 16 canales de entrada analógicos auxiliares, utilizando el multiplexor interno, tan solo consumiremos las E / S del canal dedicado VP / VN. Para seleccionar que canal auxiliar se debe convertir en cada caso, se debe colocar en la salida de la FPGA llamada MUXADDR [3:0] el canal o los canales auxiliares que se deben muestrear (Figura 40)

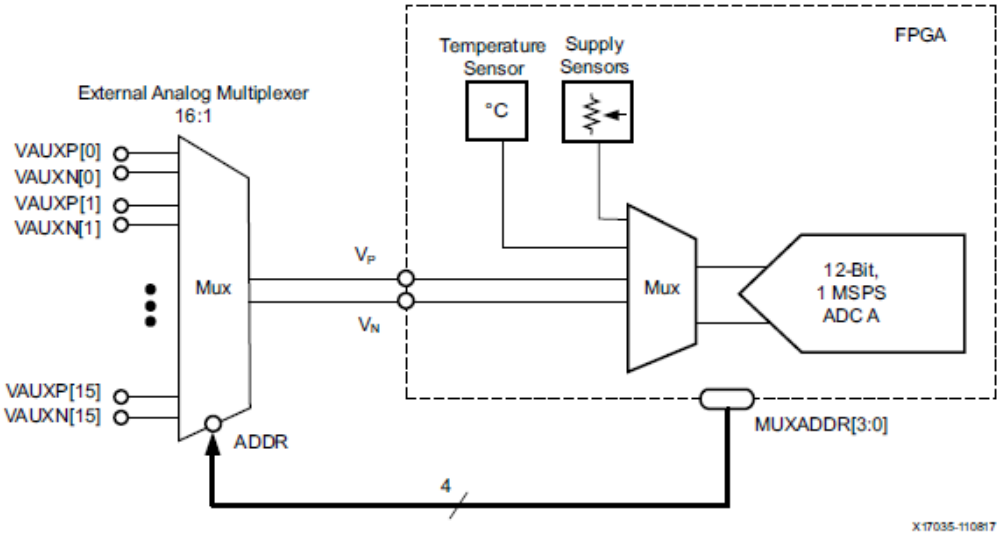


Figura 40:Modo multiplexador externo

8.1.3.6. Función de transferencia

Los resultados de las conversiones de las muestras se encuentran disponibles en los registros del 00h al 20h, tal y como se muestra en la Figura 30. Dichos registros tienen una anchura de 16 bits, aunque tan solo contienen la información los 12 bits más significativos (Figura 41).

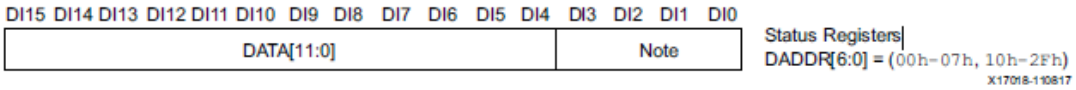


Figura 41:Valor de la conversión del XADC

Dado que existen varios modos de leer la tensión, hay que saber cómo interpretarlo según el valor obtenido en los 12 MSB de conversión.

En la configuración existe el modo unipolar, en cuyo caso acepta una tensión máxima de 1V y mínima de 0V, suponiendo cada bit un valor de 0,244 mV. En la Figura 42 se puede ver la función de transferencia para el modo Unipolar.

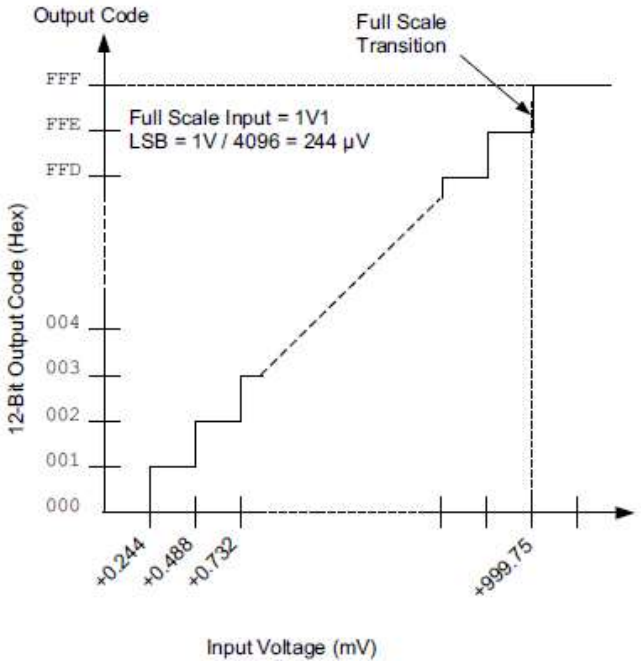


Figura 42:Función de transferencia del modo Unipolar

Además, el ADC también puede ser configurado en modo Bipolar, suponiendo una tensión mínima de -0,5V y máxima de 0,5V. También cada bit supone una diferencia de 0,244 mV. El valor mínimo de -0,5V se corresponde con un valor en hexadecimal de 800h y el valor máximo de 0,5V equivale a un 7FFh (Figura 43).

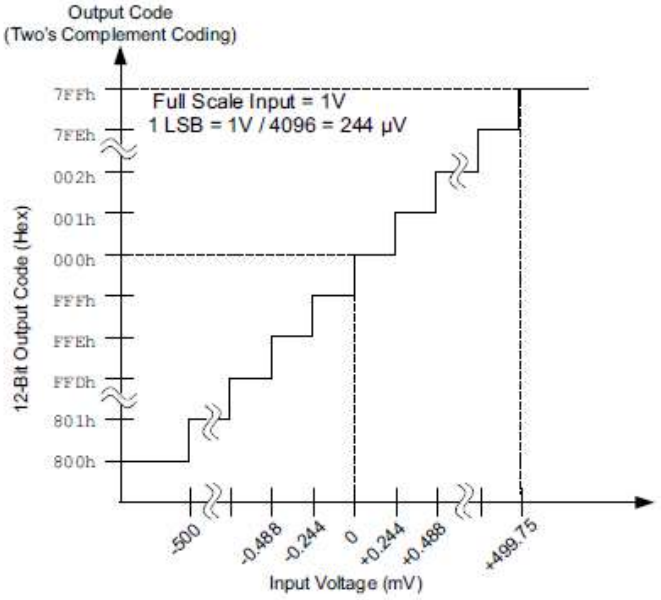


Figura 43:Función de transferencia del modo Bipolar

También se permite la lectura de la temperatura interna de la FPGA (quedando su valor en el registro 00h) y teniendo una función de transferencia que obedece a la fórmula de la Figura 44.

$$Temperature\ (°C) = \frac{ADC\ Code \times 503.975}{4096} - 273.15$$

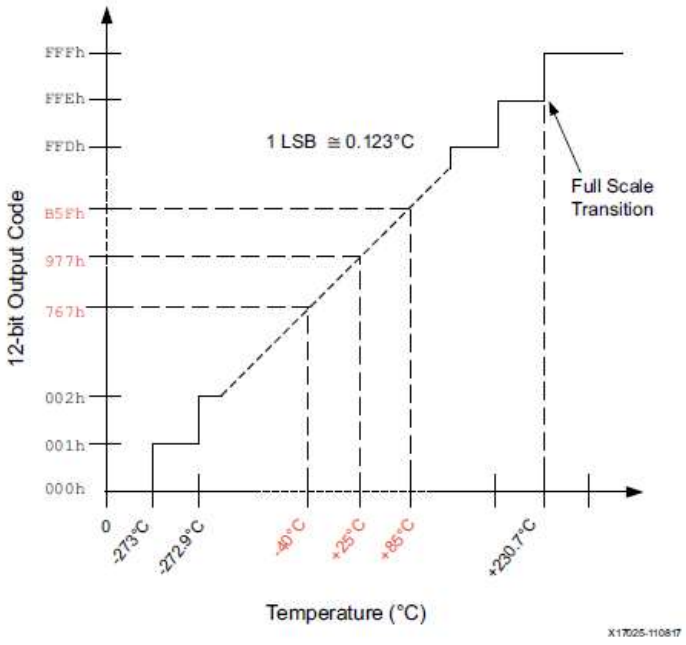


Figura 44:Función de transferencia de la temperatura interna

8.1.4. Linux

8.1.4.1. Descripción

Linux es un sistema operativo de software libre (no es propiedad de ninguna persona o empresa), por lo tanto, no es necesario comprar una licencia para instalarlo y utilizarlo en un equipo informático. Es un sistema multitarea, multiusuario, compatible con UNIX, y proporciona una interfaz de comandos y una interfaz gráfica.

Al ser software libre, el código fuente es accesible para que cualquier usuario pueda estudiarlo y modificarlo. Además de esto, este sistema cuenta con muchas distribuciones y gestores de ventanas para el entorno gráfico.

La estructura del Linux está basada en un micro núcleo híbrido que ejecuta los servicios más básicos del sistema operativo. El Kernel es el núcleo del sistema; la parte que interactúa directamente con el hardware, administrando todos los recursos de éste, como la memoria, el microprocesador, los periféricos, etc.

Además, tiene un programa que aísla al usuario del núcleo, conocido como Shell o intérprete de comandos, su función es interpretar las órdenes o aplicaciones que el

usuario mande al sistema, desde una terminal en modo texto o desde un entorno gráfico, y traducirlas a instrucciones que el sistema operativo entienda (Figura 45)

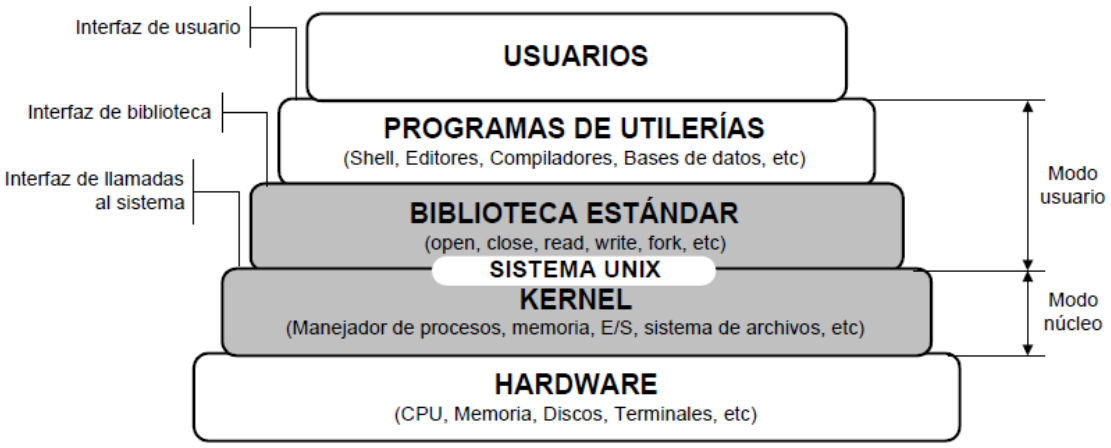


Figura 45: Estructura de Linux

Las diferentes variantes del Linux se denominan distribuciones, entre las más conocidos se encuentran Red Hat-Fedora, Suse, Debian, Ubuntu, y Mandriva.

Linux ha evolucionado mucho en los últimos años, añadiendo mejoras en las interfaces gráficas de usuario, y en el reconocimiento y utilización de los recursos hardware. Poco a poco va teniendo más usuarios y se considera una alternativa robusta y de bajo costo en comparación con los otros sistemas operativos.

En definitiva, Linux es un sistema operativo gratuito, con cada vez más prestaciones y fácilmente instalable en cualquier dispositivo electrónico. Además, se trata de un sistema operativo con gran escalabilidad, es decir, para aplicaciones que requieran elevadas prestaciones, se pueden aprovechar todas sus funcionalidades. Sin embargo, si se requiere un sistema operativo con pocas prestaciones que utilice pocos recursos de memoria, se pueden utilizar solamente las necesarias, con la posibilidad de eliminar las restantes. Debido a estas ventajas mencionadas, se ha seleccionado Linux como sistema operativo.

8.1.4.2. Xillinux

Xillinux es una distribución que incluye Linux y un kit de código FPGA y está disponible hasta el momento, para las plataformas ZedBoard, Zybo, SocKit board y MicroZed.

Está basada en Ubuntu 12.04 para ARM. Puede hacer que la tarjeta se comporte como un PC, ejecutando un entorno gráfico pudiendo conectar los mismos periféricos que a un ordenador personal. Al estar basada en Ubuntu cuenta con el soporte, comunidad y paquetes de la distribución de Linux más extendida entre sus usuarios.

El motivo principal de su elección como SO para la tarjeta, aparte de su gratuidad y su fácil manejo, es que Xillinux incluye un IP Core llamado Xillybus con el que nos podemos comunicar de forma directa o semidirecta con el hardware de la FPGA.

Además, Xillinux también incluye el device tree, u-boot y resto de herramientas necesarias para tener el sistema funcionando al instante.

8.1.4.2.1. Xillybus

Xillybus es una solución para establecer la comunicación de extremo a extremo sencilla, intuitiva y eficiente basada en DMA para el transporte de datos entre un FPGA y un host que ejecuta Linux o Microsoft Windows.

Xillybus incluye de un IP Core para la FPGA y un driver para Linux cuya misión es asegurar la comunicación PC – FPGA, sin que sea necesaria la modificación a bajo nivel de la lógica programable, ya que una vez establecida como fija, podrá ser leída, escrita o modificada desde niveles superiores como aplicaciones del sistema operativo. Para Linux, el canal de comunicación con la FPGA se ve como un fichero más del sistema, por lo que se puede escribir y leer de él con las funciones típicas de ficheros. En el otro lado, la FPGA ve la comunicación como una FIFO, lo que escriba en ella llegará al SO y lo que lea será lo que el SO le ha enviado. Es decir, Xillybus proporciona un canal de comunicación de bajo a alto nivel y viceversa (Figura 46)

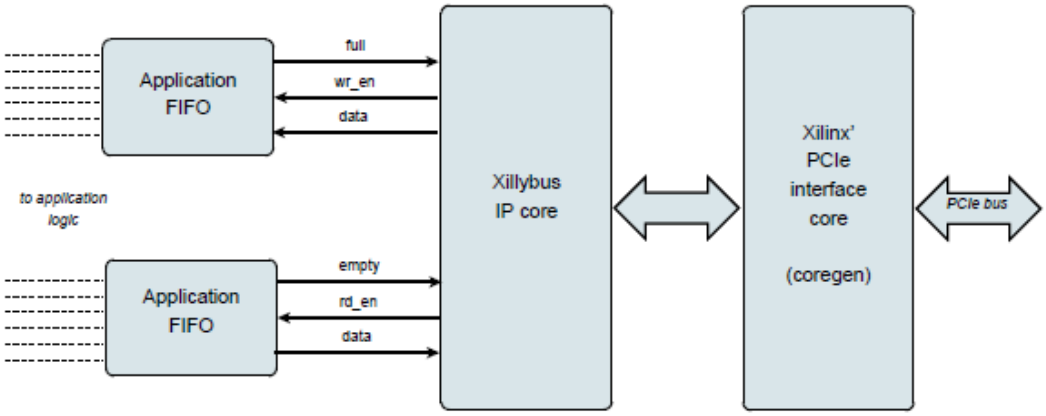


Figura 46:Comunicación FPGA (derecha) con SO (Izquierda)

Como se muestra en la Figura 46, la lógica de la aplicación en el FPGA solo necesita interactuar con las FIFOs. Por ejemplo, cuando la FPGA escribe datos en el FIFO inferior en el diagrama (FIFO de lectura), el núcleo IP de Xillybus se entera de que los datos están disponibles para su transmisión en el otro extremo de la FIFO. Por lo que, Xillybus lee los datos de la FIFO y los envía al host, lo que los hace legibles al software de usuario.

El número de flujos, su dirección y otros atributos están definidos por el usuario desde la factoría de IPs de Xillybus, disponible en el siguiente enlace:

http://xillybus.com/ipfactory/, para lograr que el ancho de banda de las FIFOs sea el adecuado en comparación con el IP de Xillybus diseñado. Por lo tanto, al diseñar las FIFOs y adjuntarlas al diseño hardware, deben iguales o en proporción al ancho de banda de Xillybus, para establecer una comunicación óptima.

Además, contiene una demo básica en la que hay implementada una comunicación entre FPGA y host para que el usuario compruebe desde el primer momento como funciona y tenga un ejemplo de cómo desarrollar su aplicación que haga uso de Xillybus.

8.1.4.2.2. Comunicación desde el lado de la FPGA al Host

El núcleo IP de Xillybus comunica los datos con la lógica del usuario a través de una FIFO estándar que es suministrado por el usuario del IP Core. Esto le da al diseñador de la FPGA la libertad de decidir la profundidad de la FIFO y su interfaz con la lógica de la aplicación.

Esta configuración libera completamente al diseñador de la FPGA de gestionar el tráfico de datos con el host. Más bien, el núcleo de Xillybus verifica las señales FIFO "vacías" (empty) o "llenas"(full) e inicia las trasferencias de datos cuando ambos extremos están listo para ello. Esta comunicación se ve reflejada en la parte inferior de la Figura 46.

8.1.4.2.3. Comunicación desde el lado del Host a la FPGA

No hay programación en el kernel o hardware necesario en el lado del host, ni ninguna necesidad de enlazar con una biblioteca de software en particular. Cualquiera puede acceder a las comunicaciones de Xillybus sin una extensión específica.

El controlador del host genera archivos de dispositivo que se comportan como de la siguiente manera (Figura 47): se abren, se leen y se escriben al igual que cualquier archivo, pero se comportan como comunicaciones entre procesos o streams TCP / IP.

La comunicación sigue la dirección de parte superior de la Figura 46.

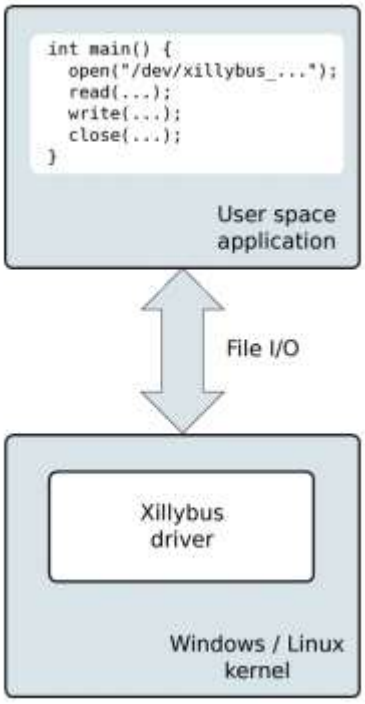


Figura 47: Comunicación entre los archivos de dispositivo

La transmisión de Xillybus está diseñada para funcionar bien con datos de alta velocidad de transferencia y bytes únicos que llegan o se envían de vez en cuando.

8.1.4.4. Características de Xillybus

Xillybus tiene unas características que lo hacen ideal para un desarrollo en una tarjeta SD portable. Algunas de estas características son las siguientes:

- Xillybus consta de un núcleo de FPGA IP Core y un controlador para la computadora en el que todo el diseño a bajo nivel ya está hecho.
- Los diseñadores de FPGA interactúan con el núcleo de IP a través de una memoria FIFO estándar o de doble puerto
- Los programadores de software a alto nivel pueden trabajar en el espacio de usuario con archivos de Linux y comunicarse con el bajo nivel de la FPGA.
- Ninguna API específica: todo el diseño del usuario se basa en métodos bien establecidos
- Robusto flujo de comunicación por línea directa que funciona correctamente.
- Hasta 160 FIFOs pueden estar entre el desarrollo del Sistema Operativo y la FPGA
- DMA utilizada exclusivamente para transferencias de datos, por lo tanto, carga mínima en el procesador.
- Baja latencia (tiempo de respuesta entre que se realiza una acción física y el dispositivo lo lleva a cabo).

- Generación personalizada de Core dependiendo de las necesidades de cada desarrollador
- Aplicaciones para las que es útil:
  - Adquisición de datos
  - Captura y reproducción de video
  - Control de FPGA desde el host
  - Fácil diseño de periféricos
  - Interfaz con hardware dedicado a una aplicación
  - Intercambio de datos con la capacidad de interpretarlos en alto nivel
- Plataformas host que lo sostienen:
  - Computadoras de PC comunes, PC-on-board (PCI) y procesadores integrados (por ejemplo, la serie i.MX de NXP)
  - Xilinx Zynq-7000
  - Intel Cyclone V SoC

8.2. Desarrollo técnico

8.2.1. Diseño inicial (Demo)

Antes de abordar el desarrollo del sistema objeto de este TFG, es conveniente comenzar a experimentar y comprobar la ejecución del sistema operativo Xillinux sobre un diseño de proyecto demostrativo. La compañía Xillybus Ltd pone a disposición un proyecto demo ya configurado en el que solo hay que generar el archivo .bit y montar todo lo necesario en la tarjeta SD. Para ello debemos ir a la página oficial de Xillybus: http://xillybus.com/xillinux/ y descargar los archivos necesarios. Como indica la Figura 48, hay dos archivos para descargar.

Figura 48: Archivos para descargar Xillybus

La primera descarga se trata de un archivo .zip que contiene 3 archivos necesarios para copiar en la tarjeta SD. Los archivos que debemos copiar son los mostrados en la Figura 49. A continuación se muestran las rutas donde se encuentran dichos archivos.

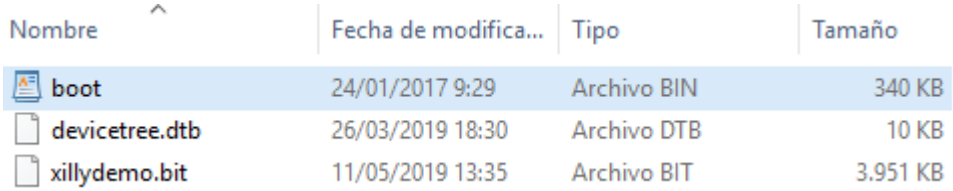


Figura 49: Archivos procedentes (sin modificar) de la descarga .zip

- **Boot.bin** (*\xillinux-eval-zedboard-2.0a\bootfiles*): se trata del gestor de arranque (bootloader). Un bootloader es un programa sencillo que no tiene la totalidad de las funcionalidades de un sistema operativo, y que está diseñado exclusivamente para preparar todo lo que necesita para iniciar el sistema operativo, en este caso Linux.
- **Devicetree.dtb** (*\xillinux-eval-zedboard-2.0a\bootfiles*): también llamado árbol de dispositivos es una estructura de datos que describe los componentes de hardware de un equipo determinado para que el kernel del sistema operativo pueda usar y administrar esos componentes, incluyendo la CPU, la memoria, los buses y los periféricos. En sistemas con un cargador de arranque que no admite árboles de dispositivos, se puede instalar un devicetree estático junto con el sistema operativo. El kernel de Linux soporta este enfoque. Los árboles de dispositivos han sido obligatorios para todos los nuevos SoC. Esto puede verse como un remedio para la gran cantidad de bifurcaciones de Linux que se han creado. El propósito es mover una parte significativa de la descripción del hardware fuera del binario del núcleo.

  Es habitual que las distribuciones de Linux basadas en Arm incluyan un cargador de arranque. Esto implica que parte del sistema operativo debe compilarse específicamente para cada tarjeta (en el caso de este proyecto se compilará para ZedBoard) o actualizarse para admitir nuevas tarjetas.

  Será necesario modificarlo como se explica en el apartado 8.2.1.3. Modificación del devicetree.dtb.
- **Xillydemo.bit**: se trata de la secuencia binaria de bits (bitstream) generada en Vivado y que contendrá el diseño hardware (puertas lógicas, máquinas de estados, etc.), escrito en HDL, del sistema que se quiere implementar en la FPGA. Para obtener este archivo es necesario abrir el proyecto xillydemo-vivado en la aplicación Vivado, hacer algunas modificaciones en VHDL y finalmente generar el bitstream x**illydemo.bit.** Para la creación de dicho archivo, es necesario seguir los pasos del apartado 8.2.1.1. Creación del Xillydemo.bit inicial

  La segunda descarga nos proporcionara la imagen para la tarjeta SD de Xillinux. Sera necesario montarlo en la SD a través de los pasos del apartado 8.2.1.2. Montaje de la imagen de Xillinux en la tarjeta SD. Tras acabar estos pasos, dentro de la tarjeta se creará un archivo binario uImage.bin.
- **uImage.bin:** se trata de la imagen binaria del kernel (núcleo) de Xillinux.

8.2.1.1. Creación del Xillydemo.bit inicial

Para crear el archivo Xillydemo.bit será necesaria la instalación de Vivado, en cualquiera de sus versiones. Tras abrir este programa, habrá que correr un Tcl (Tool Command Languaje o Herramienta de Lenguaje de Comandos).

Tcl es el lenguaje de secuencias de comandos integrado en Vivado. Tcl es un lenguaje estándar en la industria para aplicaciones de interfaces de programación.

Se debe tener claro cuál es el lenguaje HDL que se va a utilizar en el proyecto, porque dependiendo de si se usa verilog o vhdl se deberá ir a buscar el archivo .tcl a una ruta u otra dentro del .zip descargado previamente:

- En VERILOG la ruta es \xillinux-eval-zedboard-2.0a\verilog y seleccionar el archivo xillydemo-vivado.tcl
- En VHDL la ruta es \xillinux-eval-zedboard-2.0a\vhdl y seleccionar el archivo xillydemo-vivado.tcl

Tras correr el archivo .tcl (Figura 50) y seleccionarlo correctamente dentro de la carpeta /vhdl, ya que es el lenguaje que se trabajara en este proyecto (Figura 51), Vivado trabajara creando el proyecto y los directorios correspondientes. El proyecto se creará con la extensión .vhd y se guardará en la siguiente ruta: \xillinux-eval-zedboard-2.0a\vhdl\src con el nombre de xillydemo.vhd

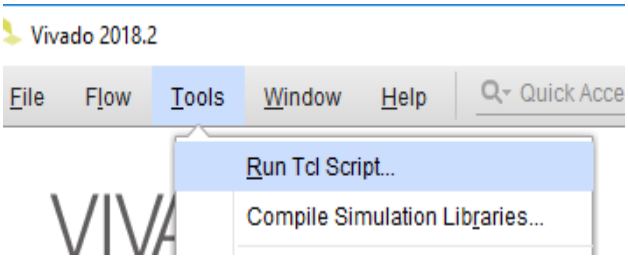


Figura 50: Selección de la opción correr Tcl Script

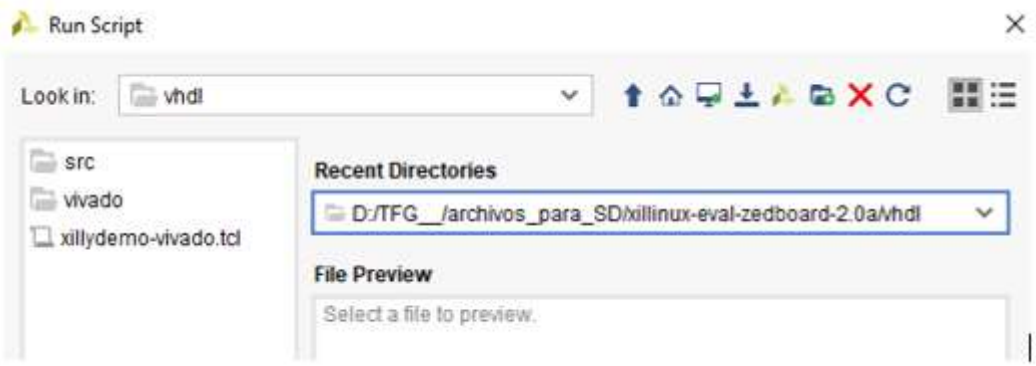


Figura 51:: Ubicación del archivo xillydemo-vivado.tcl

Es necesario hacer unas pequeñas modificaciones en archivo xillydemo.vhd antes de generar el Xillydemo.bit. Los cambios que se deben hacer son los siguientes:

- Borrar o comentar (para comentar hay que añadir “--“antes de la línea) las siguientes líneas (Figura 52), que se encuentran en el archivo de diseño superior llamado xillydemo.

```
entity xillydemo is
  port (
    -- For Vivado, delete the port declarations for PS_CLK, PS_PORB and
    -- PS_SRSTB, and uncomment their declarations as signals further below.

    --PS_CLK : IN std_logic;
    --PS_PORB : IN std_logic;
    --PS_SRSTB : IN std_logic;
    clk_100 : IN std_logic;
```

Figura 52: Líneas de entradas del puerto comentadas para Vivado

- Descomentar (borrar los dos guiones "--"que hay antes de la línea) las siguientes declaraciones de señales (Figura 53), que se encuentran en el archivo de diseño superior xillydemo

```
signal PS_CLK :  std_logic;
signal PS_PORB :  std_logic;
signal PS_SRSTB :  std_logic;
```

Figura 53: Líneas de declaración de señales descomentadas para Vivado

Tras realizar estas dos pequeñas modificaciones, ya podremos crear el archivo xillydemo.bit. Para ello, debemos clicar en la opción Generate Bitstream dentro del menú Program and Debug (Figura 54). Este proceso puede tardar varios minutos.

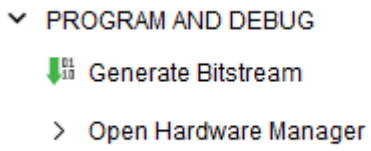


Figura 54: Opción de Generar Bitstream

Cuando termine, el archivo xillydemo.bit se encuentra en la ruta \xillinux-eval-zedboard-2.0a\vhdl\vivado\xillydemo.runs\impl_1. Este archivo es uno de los cuatro que deben estar en la SD para realizar la configuración, por lo que hay que tenerlo localizado.

Este archivo, junto con los otros tres mencionados en el apartado 8.2.1.1. Creación del Xillydemo.bit inicial forman parte de una demo de Xillinux que, entre otras cosas, nos permite manejar las FIFOs en autobucle. Es decir, lo que escribamos en una FIFO seremos capaz de leerlo de la misma. Tanto la lectura como la escritura se harán desde el Host, es decir, desde Linux (Figura 55).

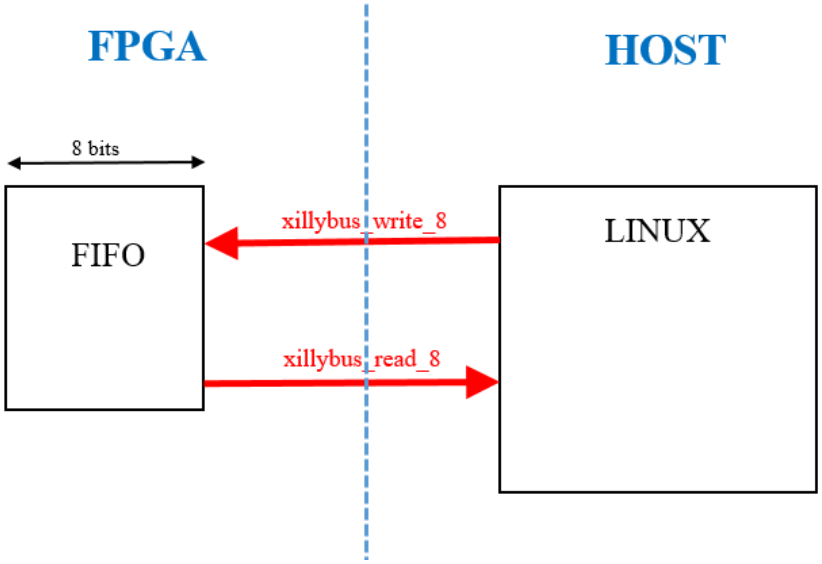


Figura 55:Comunicación FPGA-HOST desde la xillydemo.

8.2.1.2. Montaje de la imagen de Xillinux en la tarjeta SD

Para crear el archivo binario de la imagen de Xillinux, es necesario utilizar una aplicación (la que hemos utilizado es USB Image Tool), que permita crear imágenes de sistemas operativos y unidades flash USB, que se montan como unidades USB. Esta aplicación para Windows puede ser descargada desde el siguiente link: https://usb-image-tool.uptodown.com/windows. Tras su instalación, los pasos a seguir son los siguientes:

Primero hay que descomprimir el .zip de la segunda descarga (xillinux-2.0.img.gz).

Con la tarjeta SD insertada en el ordenador, se debe abrir el programa USB-Image-Tool (Figura 56)

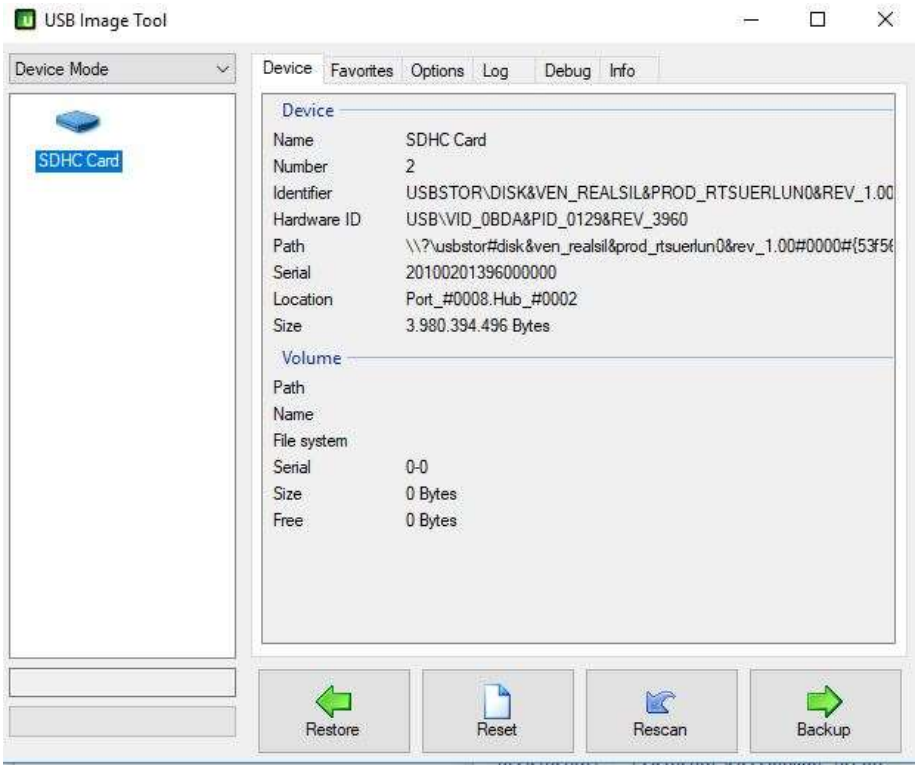


Figura 56:USB Image Tool

Dentro del Device Mode y teniendo seleccionada SDHC Card, habrá que pulsar la opción Restore y buscar el archivo que se ha descomprimido de xillinux-2.0.img.gz en el paso anterior. Este tipo de formateo lo hará en FAT32, por lo que la capacidad máxima que le dejara al sistema operativo es de 4 GBytes.

La imagen empezara a copiarse a la tarjeta SD. Este proceso puede tardar varios minutos. Al finalizar, se habrán creado dos particiones en la tarjeta SD: a la primera no se puede acceder, y la segunda será la partición de arranque en la que deben estar los cuatro ficheros mencionados en el apartado 8.2.1.1. Creación del Xillydemo.bit inicial El único archivo existente en la tarjeta cuando acabe este proceso es uImage.bin. Los otros tres archivos tendrán que ser copiados manualmente.

Es posible que Windows pida formatear la tarjeta, ya que está en un formato que no reconoce. En ningún caso se debe formatear.

8.2.1.3. Modificación del device-tree

Los procesadores ARM no tienen un BIOS que proporcione los registros iniciales y el resto de configuración, como ocurre por ejemplo en los PCs. Para que se produzca el arranque inicial, es necesaria una estructura de datos en formato de código de byte (es decir, no legible para el usuario) que contiene información útil para el núcleo al arrancar. El cargador de arranque accede a esa porción de datos en una dirección conocida en la RAM antes de saltar al punto de entrada del kernel. Esa estructura es el árbol de dispositivos o device-tree.

La adopción de la estructura de árbol estándar permite utilizar una API conveniente para obtener datos específicos. Por ejemplo, hay una convención clara y precisa sobre cómo definir periféricos en el bus y una API para obtener la información esencial que necesita el conductor: direcciones, interrupciones y variables personalizadas. El árbol de dispositivos es donde informamos al núcleo acerca de una pieza específica de hardware (es decir, la lógica PL) que hemos agregado o eliminado, para que el núcleo pueda comenzar con el controlador correcto para manejarlo (o abstenerse de hacerlo si el hardware fue eliminado). Aquí también es donde se transmite información específica sobre el hardware.

Es necesario modificar el device-tree tal y como se va a indicar a continuación para eliminar la función que viene por defecto para controlar el XADC, ya que no nos interesa la que viene, sino agregar la que se ha creado en el archivo xillydemo .bit. Si se usa el devicetree.dtb que incluye el kit de Xillybus, sin modificar, aunque todo se encuentre bien configurado, los valores que devuelve el XADC son erróneos ya que provienen de un canal inválido.

Para la modificación será necesario disponer del sistema operativo Linux en un PC o en una máquina virtual (no tiene por qué ser Xillinux). La instalación de un sistema operativo en un PC se puede ver en el apartado 8.2.5. Instalación de Linux en un ordenador

Lo primero que hay que hacer es instalar un compilador que permite convertir el archivo .dtb (extensión device-tree) a un archivo .dts (extensión archivo de datos) que si puede ser modificado por un editor de textos corriente. Para ello, abriendo una terminal en Linux (Cntl+Alt+T), habrá que escribir el siguiente comando:

```
sudo apt-get install device-tree-compiler
```

Copiando el device-tree del kit inicial de Xillinux, en un directorio localizado, por ejemplo, el Escritorio y situándose en ese directorio, se debe ejecutar el siguiente comando:

```
dtc –I dtb –O dts –o devicetree.dts devicetree.dtb
```

Este comando tiene como significado:

```
dtc –I <formato de entrada> -O <formato de salida> -o
    <fichero de salida> <fichero de entrada>
```

Tras obtener el fichero devicetree.dts, se puede abrir y modificar con un editor de textos. Hay que buscar las siguientes líneas de código y eliminarlas:

```
ps7-xadc@f8007100 {
clocks = <0x3 0xc>;
```

```
compatible = "xlnx,ps7-xadc-1.00.a";
interrupt-parent = <0x2>;
interrupts = <0x0 0x7 0x4>;
reg = <0xf8007100 0x20>;
};
```

Una vez eliminado dicho código, hay que guardar el fichero y hacer la operación inversa, es decir, convertir el archivo .dts en archivo .dtb mediante el siguiente comando:

```
dtc -I dts -O dtb -o devicetree.dtb devicetree.dts
```

Los comandos de ejecución desde la terminal de Linux se pueden observar en la Figura 57.

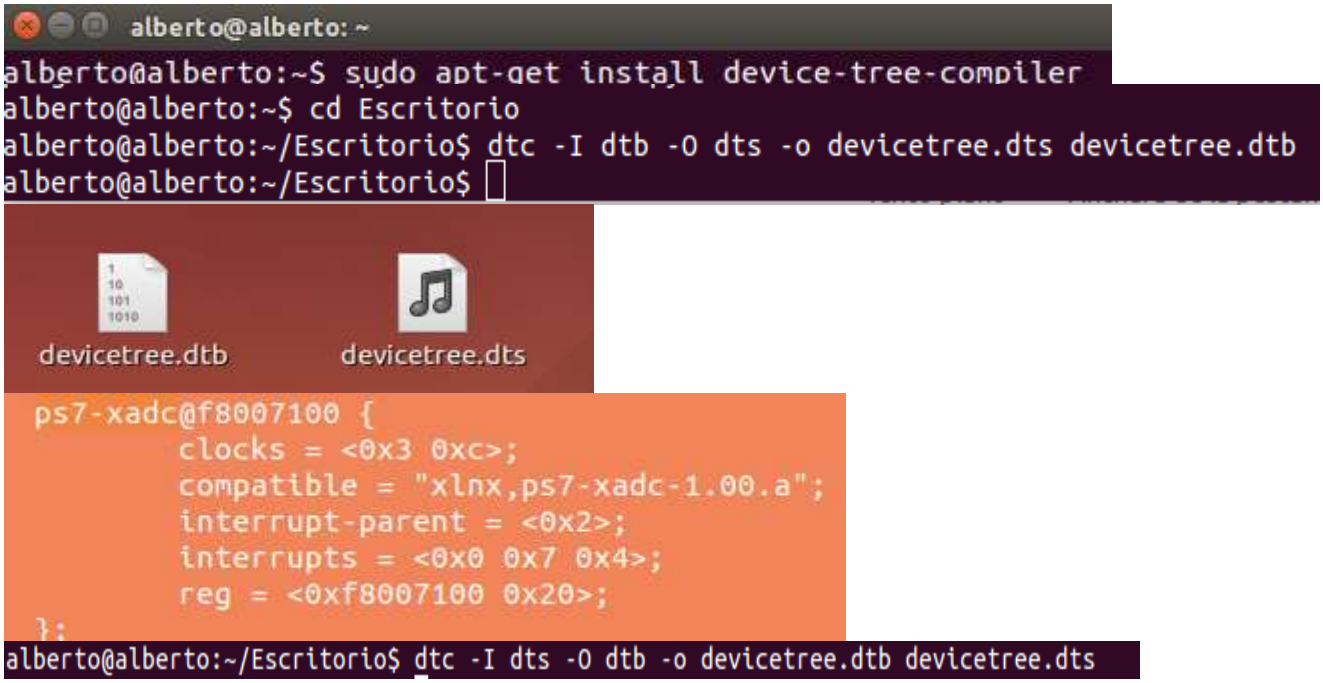


Figura 57:Pasos en la modificación del devicetree.dtb

Por lo tanto, el archivo devicetree.dtb ya modificado será el que ira situado en la tarjeta junto con la imagen uImage, el xillydemo.bit y el boot.bin (Figura 58)

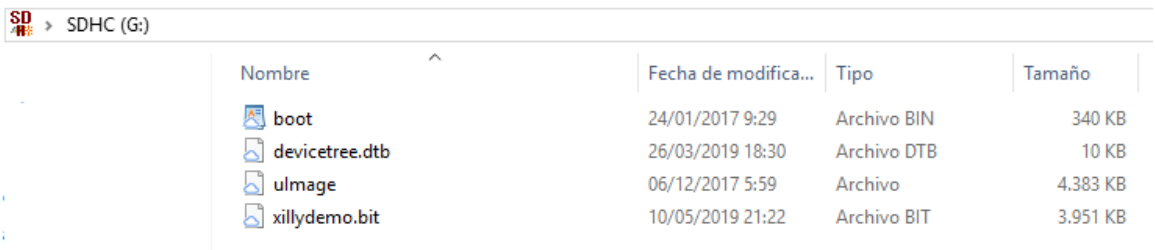


Figura 58:Archivos necesarios en la tarjeta SD

8.2.2. Montaje Inicial (Demo)

Tras copiar los cuatro archivos mencionados en la Figura 8.2.1.3.2 será necesario configurar los jumpers de la tarjeta ZedBoard de forma que el arranque sea desde SD y, además, también lea un archivo .bit. Este tipo de configuración está expuesta en el apartado 8.1.2.4. Modos de configuración. Hay que poner a nivel alto los pines JP10, JP9 y cortocircuitar el JP6 (Figura 59).

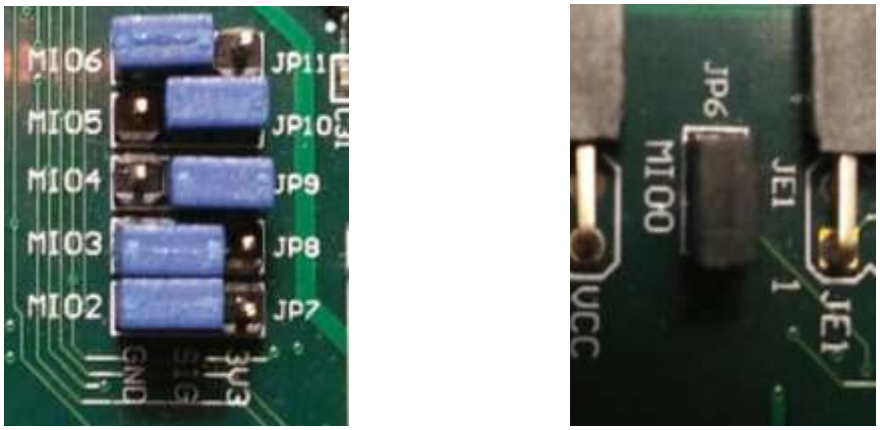


Figura 59: Conexiones de los pines para arranque leyendo la SD y el bitstream

Dado que Xillinux maneja un monitor mediante la salida VGA, habrá que conectar una pantalla a través de este conector. Para completar la interfaz se puede conectar un teclado y un ratón mediante el puerto USB (Figura 60). De este modo, la interfaz de usuario será prácticamente igual que la de un ordenador. Además, dado que la ZedBoard cuenta con entrada de conexión Ethernet, también se le puede conectar directamente de un router. Permitir que se pueda conectar a internet va a ser necesario a la hora de instalar librerías en el sistema operativo, para que sea capaz de leer la interfaz gráfica de usuario GUI (apartado 8.2.6. Creación de la aplicación GUI)

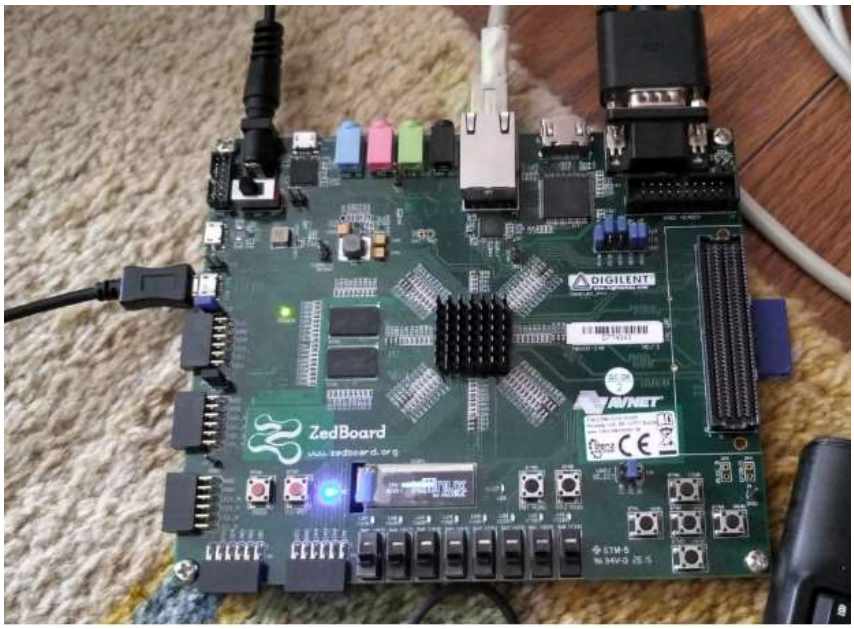


Figura 60: Conexiones externas de la ZedBoard (alimentación, USB, VGA y Ethernet)

Xillinux cuenta con una interfaz gráfica X-window, que se activa (comienza a ejecutarse), escribiendo el comando:

```
startx
```

La interfaz gráfica que muestra Xillinux es la siguiente (Figura 61):

Figura 61:Interfaz gráfica de Xillinux

Para comprobar el funcionamiento de las FIFOs, se escribirá en la FIFO de escritura xillybus_write_8 y dado que está conectada en autobucle, se deberá leer en la FIFO de lectura xillybus_read_8. Para ello, desde Xillinux de la FPGA, se abrirán dos terminales, uno para leer y otro para escribir.

- Para escribir se utiliza el comando:

```
cat > /dev/xillybus_write_8
```

texto de prueba

- Para leer se utiliza el comando:

```
cat / dev/xillybus_read_8
```

(se debe leer el mismo texto de prueba que se ha enviado desde el host a la FIFO de escritura)

Como se observa en la Figura 62, la prueba inicial para comprobar el funcionamiento en autobucle de las FIFOs funciona, con el flujo de datos que indica la Figura 55.

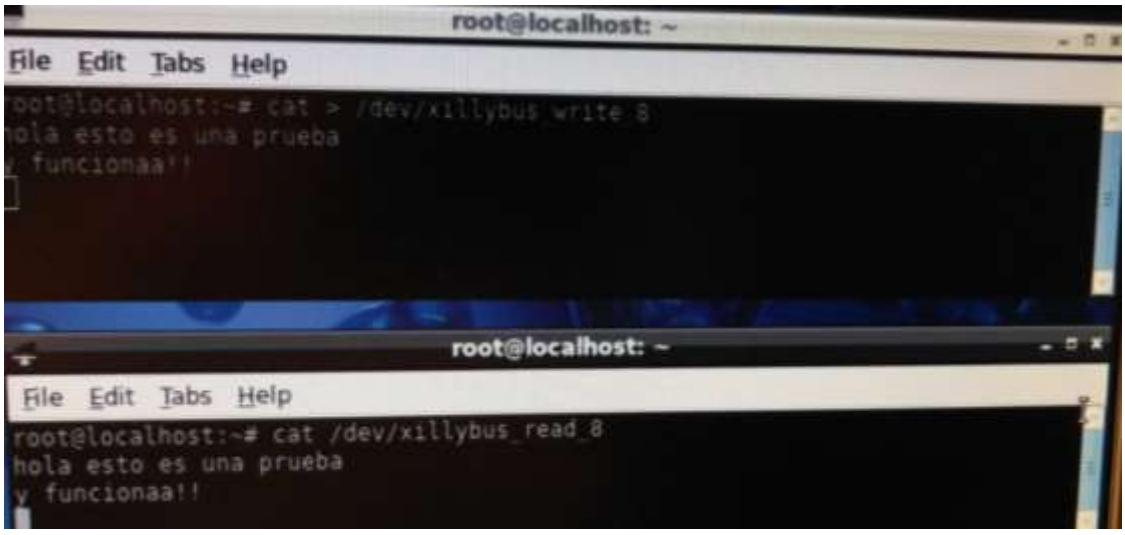


Figura 62:Arriba: se escribe en la FIFO de escritura. Abajo: se lee de la FIFO de lectura el mismo texto que se ha escrito.

Por lo tanto, con estas pruebas iniciales se puede comprobar que la demo inicial de Xillinux es operativa. Ahora, llegados a este punto, es necesario realizar un conjunto de modificaciones conforme a los requerimientos y esquema de bloques que se requiere en nuestro proyecto (apartado 8.2.3. Esquema general del sistema. Bloques a desarrollar).

8.2.3. Esquema general del sistema. Bloques a desarrollar.

Para nuestro sistema *Datalogger* a desarrollar, será necesario contar con una estructura principal basada en bloques, como la que se muestra en la siguiente Figura 63.

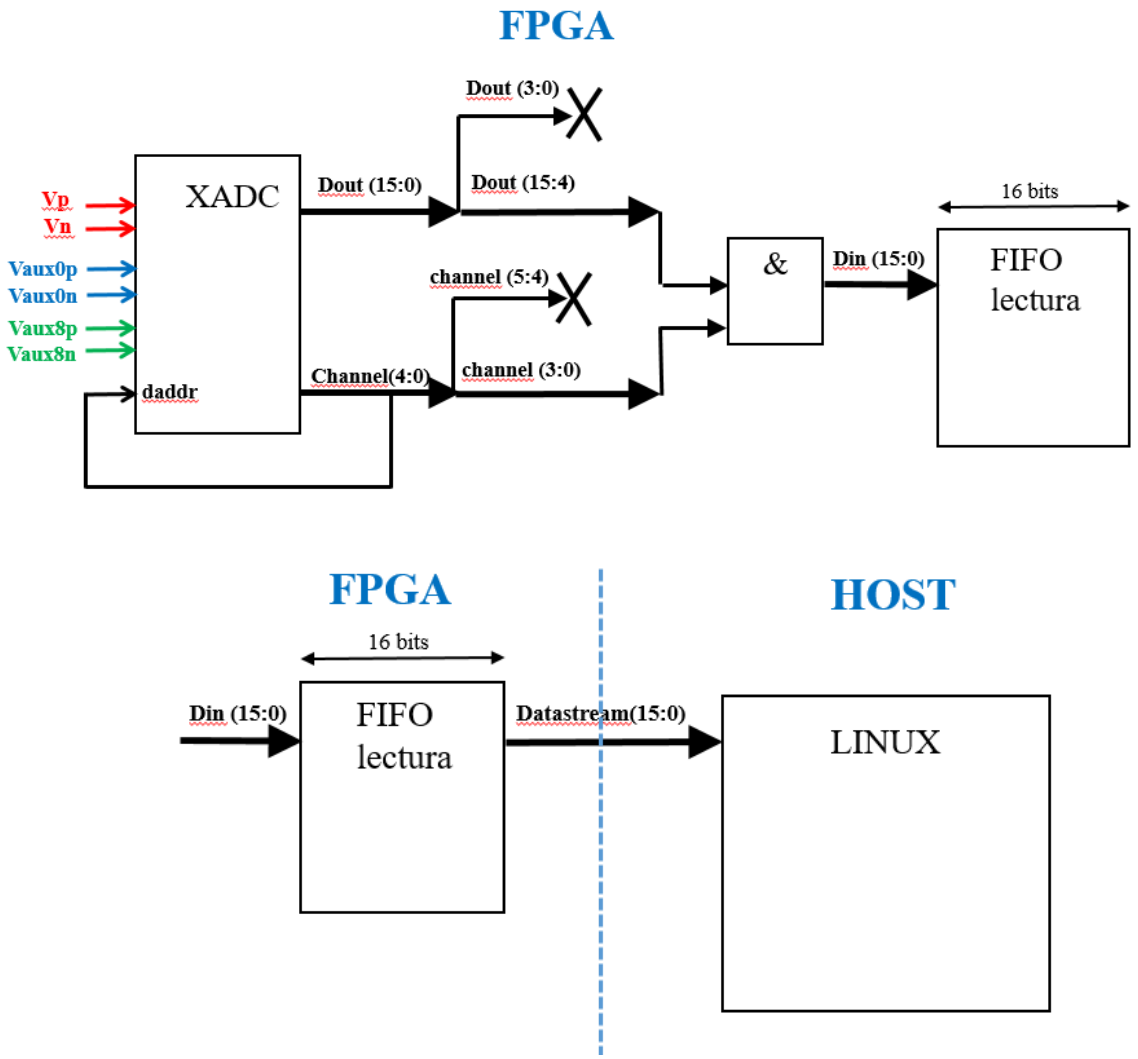


Figura 63:Esquema de bloques principal

Este esquema está formado por varios bloques que se explicarán a continuación:

- **XADC**: para captar las señales analógicas de tres canales: el canal dedicado VPVN, el canal auxiliar Vaux0 y el canal auxiliar Vaux8.

  El XADC está configurado en modo Secuencia Continua, así que la captura de datos se ve en la Figura 35:Muestreo Continuo. Como la conversión de datos está formada por dos fases (adquisición y conversión), el funcionamiento es el siguiente: primero ocurre la adquisición del canal VPVN en un cierto periodo de tiempo. Tras finalizar ese periodo, los datos adquiridos del canal VPVN pasan a la fase de conversión en la que son transformados a 16 bits. Mientras estos datos se convierten, ocurre la adquisición del canal Vaux0 en el mismo periodo de tiempo. Tras finalizar este tiempo, los datos adquiridos del canal Vaux0 pasan a la fase de conversión, y a su vez, se adquieren los del canal Vaux8. Cuando concluye el tiempo, se produce la fase de conversión del canal Vaux8, mientras se adquieren los datos del canal VPVN. Este bucle se repite de forma indefinida mientras esté encendido el dispositivo. Es decir, se trata de un proceso de conversión S/H (Sample and Hold) en el que un

condensador se queda cargado durante un cierto tiempo y posteriormente se convierte dicho valor analógico.

Dado que el dato de salida del conversor es de 16 bits y solo los 12 MSB son los que nos dan información sobre la conversión, los 4 bits LSB que no contienen información relevante, pueden ser ignorados/sustituidos por, por ejemplo, la identificación del canal. Por lo tanto, la FIFO que se necesita debe ser de 12 bits (para que el dato de la conversión pueda ser tomado entero), más los bits añadidos de referencia del canal (por ejemplo 4 bits).

- **FIFO:** Se utilizará una FIFO de 16 bits. Los datos de entrada de esta FIFO de lectura son (Tabla 9):

  - 12 MSB de la salida dout de conversión

  - 4 LSB de la salida channel del XADC

Salida XADC	Entrada FIFO
XADC dout (15:4)	FIFO (15:4)
XADC dout (3:0)	Open (sin conectar)
XADC channel (4)	Open (sin conectar)
XADC channel (3:0)	FIFO (3:0)

Tabla 9: Salida del XADC y entrada a la FIFO

Los 16 bits de salida de la FIFO están conectados con los 16 bits de entrada al host Xillinux, mediante la entrada xillybus_datastream.

- **HOST (Linux):** A través del cual se podrán manejar todos los datos que haya recibido. El host está conectado con la FIFO en su entrada xillybus_datastream, por lo que se podrá acceder a estos datos en la ruta /dev/xillybus_datastream, y aparecerán un flujo de datos como se muestra en la Figura 88.

  El Host también permite la opción de enviar un flujo de datos a través del controlstream. En el apartado 8.2.3.4. Creación del IP de Xillybus se muestra como se ha creado dos flujos de datos: datastream (datos de entrada al Host) y controlstream (datos de salida del host). El datastream ya se ha explicado que su funcionamiento es el de recibir un flujo de datos de 16 bits de entrada al Host. El controlstream tiene la función de enviar datos de salida del Host a la FPGA. En este caso no se ha conectado a ninguna señal, es decir, no se utiliza. Pero en caso de querer utilizarlo, habría que conectarlo con las señales correspondientes. El controlstream enviara flujos de datos de 8 bits (porque así se ha establecido en el apartado 8.2.3.4. Creación del IP de Xillybus) y se puede acceder a él mediante la ruta de Linux /dev/xillybus_controlstream.

En los siguientes apartados se procederá a configurar y conectar cada una de las partes del esquema.

8.2.3.1. Creación del bloque XADC

Dado que Vivado cuenta con varios IP Cores (módulos de propiedad intelectual), para la creación e instanciación del XADC, se utilizará uno de ellos. Además, cuenta con un asistente para configurarlos según las necesidades que requiera el proyecto.

Para la creación del IP Core del XADC hay que seguir los siguientes pasos:

En el menú Project Manager, seleccionar la opción IP Catalog (Figura 64)

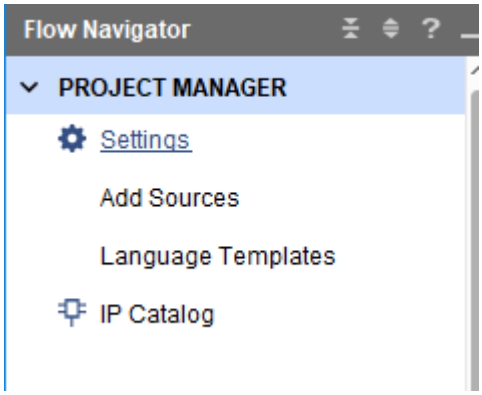


Figura 64:Selección de IP Catalog

Se abrirá un catálogo como el que se muestra en la Figura 65, en el que se debe escribir “xadc” y seleccionar la opción XADC Wizard.

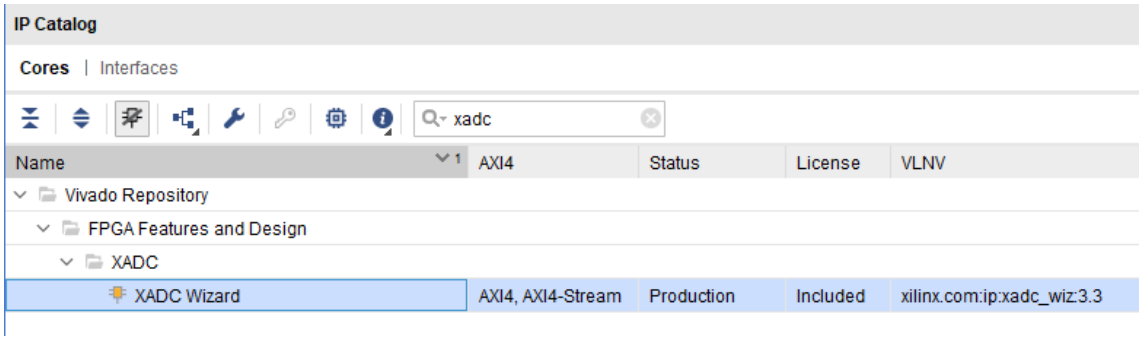


Figura 65: Catalogo de IPs con XADC Wizard seleccionado

A continuación, se abrirá un asistente que permite configurar el XADC (Figura 66). Para seleccionar las opciones más interesantes de acuerdo al proyecto, se recomienda ir al apartado 8.1.3. XADC.

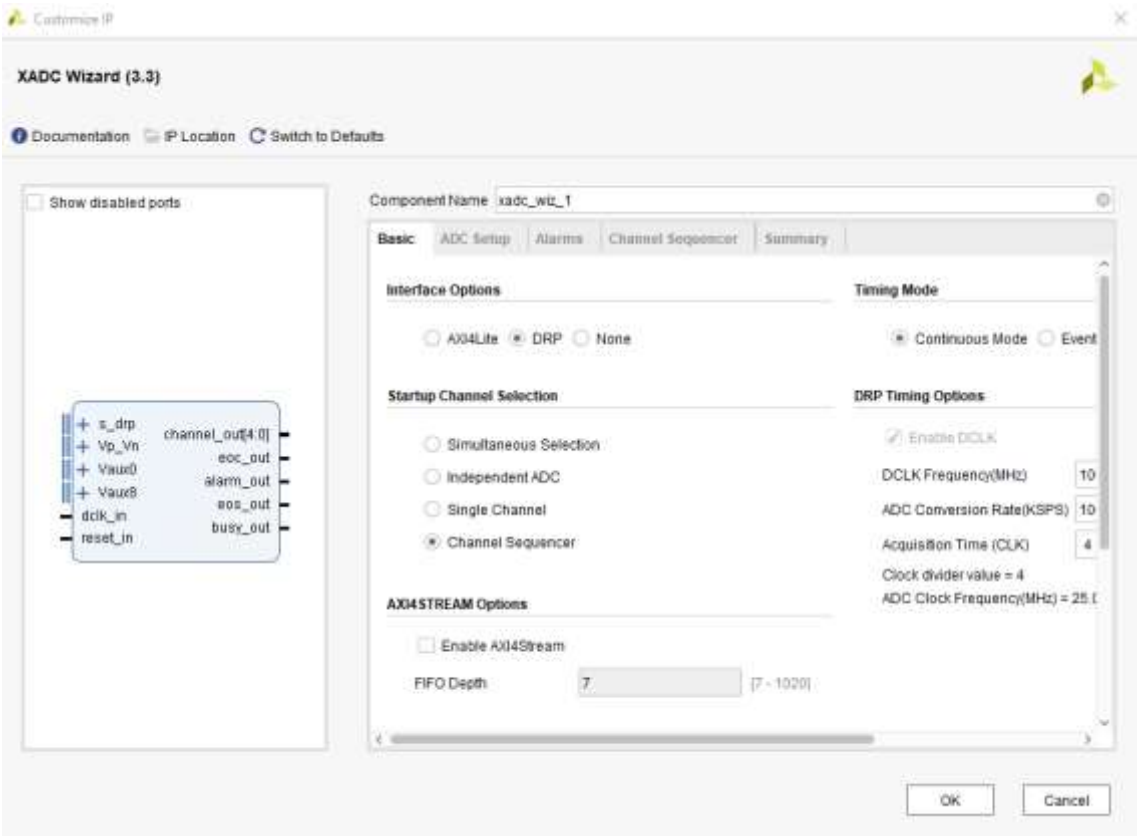


Figura 66: Asistente para la creación de IP XADC

En la pestaña Basic encontramos varias opciones de configuración:

- **Interface Options**: se debe seleccionar DRP para que el XADC utilice únicamente dicha opción, sin utilizar las LUTs
- **Startup Channel Selection**: se seleccionará la opción Channel Sequencer para que muestree los tres canales de forma continua e ininterrumpida
- **AXI4STREAM Option**: no se debe seleccionar ninguna opción de este asistente. El propio XADC cuenta con una FIFO donde almacenar datos de las muestras tomadas. Por defecto, viene el valor mínimo 7. Se dejará este valor ya que no se utilizará esta FIFO en el proyecto, sino que se utilizar una externa conectada donde irán los datos de las capturas.
- **Timing Mode**: se selecciona Continuous Mode para las muestras las tome de forma continua e ininterrumpida y haga su correspondiente conversión. De esta forma no tendrá que esperar a ningún evento de disparo.
- **Control/Status Ports**: por defecto viene activada la opción reset_in, que se dejara activada para que se pueda resetear el XADC cuando sea necesario. Además, se activará la opción JTAG Arbiter, que creará tres salidas del ADC: jtaglocked_out, jtagbusy_out y jtagmodified_out. No se utilizarán estas salidas en el proyecto, pero en el caso de que se quieran realizar pruebas o simulaciones mediante el puerto JTAG, estas salidas serian de gran utilidad.
- **DRP Timing Options**:

- **DCLK Frequency (MHz):** será la frecuencia a la que irá la señal de reloj DRP. Se utilizará la frecuencia de 100MHz que proporciona Xillybus.
- **ADC Conversion Rate (KSPS):** Teniendo en cuenta la existencia de dos relojes para el sistema: el reloj DCLK (para el DRP) y el reloj ADCCLK (para el funcionamiento interno del XADC)

  El ADCCLK debe ser de una frecuencia inferior al DCLK, ya que se divide por un factor llamado Clock divider value y que depende de la división de ambos relojes.

  El reloj ADCCLK debe funcionar a unas velocidades de entre 1MHz y 26 MHz. Por lo tanto, el mínimo valor que se le puede dar al ADC Conversion Rate (KSPS) es de 40KSPS para que el reloj de ADCCLK funcione a la mínima velocidad, que será de 1,03MHz (siendo el divisor Clock divider value de 97). Se muestra la configuración en la Figura 67.

  Se ha elegido la menor velocidad de muestreo para el XADC dado que al tratarse de este tipo de aplicación, no se necesita que las muestras sean tomadas a alta velocidad. Las variables externas que se quieren muestrear no variarán significativamente en un tiempo pequeño, Por ejemplo, si una variable externa va a ser la temperatura atmosférica, ésta no va a cambiar significativamente en el transcurso de un segundo.

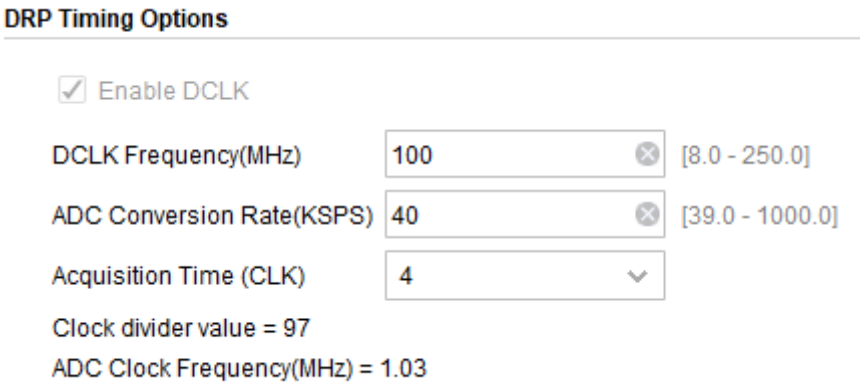


Figura 67:DRP Timing Options

- **Acquisiton time (CLK)**: esta opción se deshabilita automáticamente al seleccionar la opción de Channel Sequencer.
- **Analog Sim File Options**: dado que no se va a realizar ninguna simulación, esta opción se dejará como está por defecto

En la pestaña ADC Setup existen varias opciones que hay que configurar:

- **Sequencer Mode**: debe seleccionarse la opción Continuous, para que el conversor recorra la secuencia de canales seleccionados de forma continua
- **Channel Averaging**: Se utilizará la opción de hacer una media de 16 muestras, es decir, el conversor XADC tomara 16 muestras y nos proporcionara un valor

que es el promedio de esas 16 muestras. Hay que tener en cuenta que cuanto mayor sea el número de muestras con las que hace la media, tendrá mayor precisión, pero menor velocidad de muestreo, ya que para que el conversor proporcione un valor, tendrá que tomar 16, 64 o 256 muestras.

Se utiliza este promediado ya que no se necesita que el conversor sea especialmente rápido y posteriormente, por software, se realizara otro promediado, por lo el valor proporcionado será más preciso.

- **ADC Calibration**: se marcarán ambas opciones "ADC Offset Calibration" y "ADC Offset and Gain Calibration". Esto permite corregir los errores de calibración y de esta manera impedir que el conversor se sature antes de tiempo.
- **Supply Sensor Calibration**: al igual que en el caso anterior, se marcarán ambas opciones "Sensor Offset Calibration" y "Sensor Offset and Gain Calibration", para impedir que el sensor se sature en un momento indeseado.
- **External Multiplexer Setup**: esta opción se dejará desactivada, ya que no se utilizará el multiplexor externo, sino que se recogerán los datos de los registros de los canales de entrada analógica. Para más información se recomienda mirar el apartado 8.1.3.5. Modos de operación
- **Power Down Options**: esta opción se dejará desactivada como viene por defecto

En la pestaña Alarms, se desactivarán todas las alarmas disponibles. Como se explica en el apartado 8.1.3.3.2. Registros del XADC, estas alarmas nos permiten ajustar los umbrales de temperatura, Vcc, etc., a los que es conveniente que avise. En este caso, no es interesante que salte ninguna alarma.

La pestaña Channel Sequencer nos permite ajustar los canales que el XADC debe muestrear, además de poder habilitar la el promediado de muestras con el valor puesto anteriormente y activar o no el modo bipolar (Figura 68). Para entender bien dicho modo se recomienda mirar el apartado 8.1.3.6. Función de transferencia, ya que, utilizando esta opción, la tensión que soporta el ADC va desde -0,5V a +0,5V. Sin embargo, si no se activa esta opción, se habilita el modo unipolar, cuya tensión de entrada oscila entre 0V y 1V.

	Channel Enable	Average Enable	Bipolar	Acquisition Time
VP/VN	☑	☑	☑	☐
vauxp0/vauxn0	☑	☑	☑	☐
vauxp8/vauxn8	☑	☑	☑	☐

Figura 68:Selección de canales en la pestaña Channel Sequencer

Para finalizar, la última pestaña Summary nos muestra un resumen de cómo va a quedar configurado el XADC, para comprobar que se ha configurado correctamente (Figura 69)

Summary

Interface Selected	DRP
XADC operating mode	channel_sequencer
AXI4Stream Interface	false
Timing Mode	Continuous
DCLK Freq(MHz)	100
Sequencer Mode	Continuous
Channel Averaging	16
Enable External Mux	false

Figura 69: Resumen del asistente del XADC

Para resumir, la función de este asistente es modificar los registros del XADC. Estos registros también pueden ser modificados en HDL, a través de un macro con valores de inicialización o en ejecución a través del interfaz DRP escribiendo en los registros los valores que se requieran. Para ello, habría que consultar el apartado 8.1.3.2. Registros del XADC.

Al finalizar la configuración se creará un archivo llamado xadc_wiz_0.vhd que tendrá que ser conectado con el proyecto final. Se trata de un archivo wrapper (envoltorio del bloque) en el cual se encuentran las entradas/salidas y configuraciones internas.

El wrapper xadc_wiz_0.vhd va a ir contenido en otro fichero, también con extensión .vhd que será el que se conecte con el proyecto final. De esta forma eliminaremos entradas y salidas innecesarias y se harán algunos cambios necesarios para el proyecto.

8.2.3.2. Creación del top_xadc.vhd

Lo primero que hay que hacer es crear un fichero nuevo con la extensión .vhd. Para ello hay que clicar en el menú Project Manager, en el apartado Add Sources, y después en Add or create design sources. Tras pulsar Next, pulsando en el botón Create File, nos aparece un asistente para elegir el tipo, el nombre y la localización.

- File Type: VHDL
- File name: top_xadc
- File location: <Local to Project>

Los puertos de entrada y salida no se definirán desde el asistente, sino que se harán posteriormente.

En la siguiente Figura 70 se muestran los pasos para la creación del top_xadc.vhd

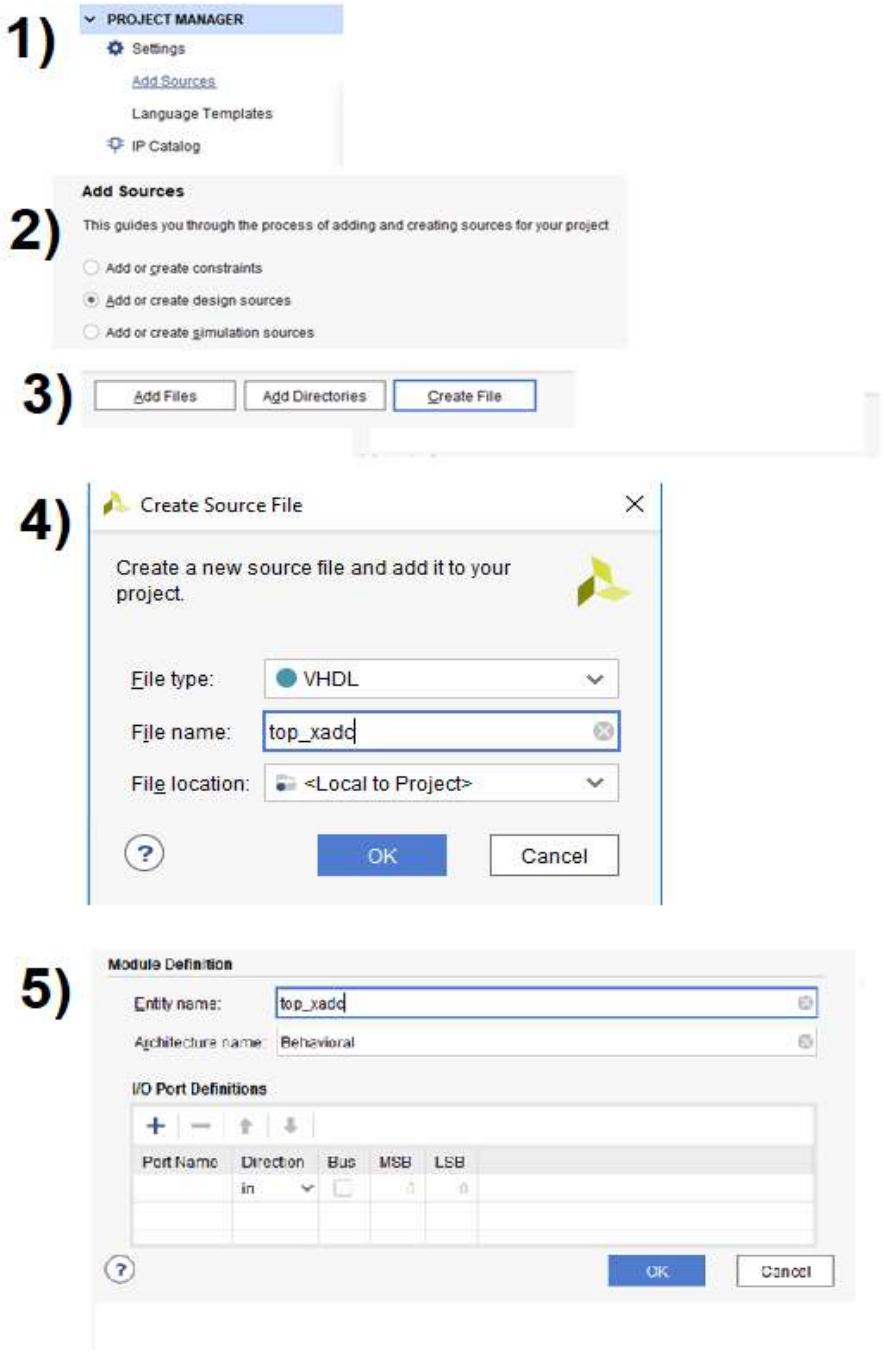


Figura 70: Pasos para la creación del top_xadc.

Lo primero de todo es crear el puerto de entradas y salidas del top_xadc que se conectaran a las FIFOs y al sistema en el proyecto final. Dicho puerto se encuentra reflejado en la Figura 71.

```
entity top_xadc is
Port (clk : IN std_logic;
reset : IN std_logic;
vp_in : IN STD_LOGIC;
vn_in : IN STD_LOGIC;
vauxp0 : IN STD_LOGIC;
vauxn0 : IN STD_LOGIC;
vauxp8 : IN STD_LOGIC;
vauxn8 : IN STD_LOGIC;
jtagLMB : OUT std_logic_vector(2 downto 0);
dout : OUT std_logic_vector(15 downto 0);
drdy : OUT std_logic;
channel : OUT std_logic_vector(4 downto 0)
);
end top_xadc;
```

Figura 71:Puerto de entradas y salidas del top_xadc

Una vez creado, hay que editarlo y añadirle los componentes y las señales adecuadas. Para ello hay que instanciar el componente xadc_wiz_0 creado en el apartado anterior. Entre la declaración de la arquitectura y la señal de comienzo “begin” hay que copiar el port de la siguiente manera (Figura 72). Además, se han creado señales auxiliares para conectar las entradas y salidas del xadc_wiz_0.

```
architecture Behavioral of top_xadc is

component xadc_wiz_0 is
port ( daddr_in : in STD_LOGIC_VECTOR (6 downto 0);
den_in : in STD_LOGIC;
di_in : in STD_LOGIC_VECTOR (15 downto 0);
dwe_in : in STD_LOGIC;
do_out : out STD_LOGIC_VECTOR (15 downto 0);
drdy_out : out STD_LOGIC;
dclk_in : in STD_LOGIC;
reset_in : in STD_LOGIC;
jtagbusy_out : out STD_LOGIC;
jtaglocked_out : out STD_LOGIC;
jtagmodified_out : out STD_LOGIC;
vauxp0 : in STD_LOGIC;
vauxn0 : in STD_LOGIC;
vauxp8 : in STD_LOGIC;
vauxn8 : in STD_LOGIC;
busy_out : out STD_LOGIC;
channel_out : out STD_LOGIC_VECTOR (4 downto 0);
eoc_out : out STD_LOGIC;
eos_out : out STD_LOGIC;
alarm_out : out STD_LOGIC;
vp_in : in STD_LOGIC;
vn_in : in STD_LOGIC
);
end component;
-- señales auxiliares
signal aux_dout : std_logic_vector(15 downto 0);
signal aux_channel : std_logic_vector(4 downto 0);
signal aux_eoc : std_logic;
signal aux_daddr : std_logic_vector(6 downto 0);
signal senal_cortada: std_logic_vector (15 downto 0);

begin
```

Figura 72:Instanciación del componente xadc_wiz_0 y creación de señales auxiliares.

Tras el "begin", se pueden hacer las conexiones del componente xadc_wiz_0 con las diferentes señales auxiliares y las entradas y salidas del top_xadc, quedando como indica la Figura 73.

```
xadc : xadc_wiz_0
port map (
daddr_in => aux_daddr,
den_in => aux_eoc,
di_in => (others => '0'),
dwe_in => '0',
do_out => aux_dout,
drdy_out => drdy,
dclk_in => clk,
reset_in => reset,
jtagbusy_out => jtagLMB(0),
jtaglocked_out => jtagLMB(2),
jtagmodified_out => jtagLMB(1),
vauxp0 => vauxp0,
vauxn0 => vauxn0,
vauxp8 => vauxp8,
vauxn8 => vauxn8,
busy_out => open,
channel_out => aux_channel,
eoc_out => aux_eoc,
eos_out => open,
alarm_out => open,
vp_in => vp_in,
vn_in => vn_in
);
```

Figura 73: Conexiones del xadc_wiz_0 dentro del top_xadc.

Además, hay que hacer ciertas modificaciones para que los datos salgan de forma adecuada. A la dirección de lectura del xadc “daddr” hay que conectarle la salida “channel”. Como “channel” es de 5 bits y “daddr” es de 7 bits, los dos MSB tienen que ser “00”.

La salida del conversor “dout” son 16 bits, de los cuales los 4 LSB son despreciables, por lo que se pueden quitar. En el lugar de estos 4 bits se colocarán los 4 LSB de la salida “channel” para indicar en cada momento la conversión de que canal se está realizando.

Además, Xillybus invierte el orden de los bytes, por lo que es necesario cambiar el orden previamente. Todos estos cambios se ven reflejados en la Figura 74.

```
-- puesto que daddr es de 7 bits y channel es de 5,
-- los 2 primeros bits serán siempre 0
aux_daddr <= "00" & aux_channel;
-- xillybus invierte el orden de los bytes, por eso
-- hay que poner primero el byte menos significativo
-- hasta el más significativo
senal_cortada<=aux_dout(15 downto 4) & aux_channel(3 downto 0);
dout <= senal_cortada(7 downto 0) & senal_cortada(15 downto 8);
-- se conecta la señal
channel <= aux_channel;
```

Figura 74: Adecuación de salida de los datos en el top_xadc

8.2.3.3. Creación de la FIFO de lectura

La FIFO se trata de un buffer de almacenamiento donde se van guardando en orden los datos que hayan sido captados por el XADC. El primer dato en entrar va a ser el primer dato en salir. Dado que la salida de datos del bloque top_xadc es de 16 bits (12 bits del valor y 4 bits del canal), es necesario que el ancho de la FIFO tenga como mínimo esos 16 bits, para que no se pierda ningún bit.

El proceso de creación de la FIFO es muy similar al del XADC, ya que se hará mediante el catálogo de IP (Figura 75). En lugar de escribir xadc, como en el caso anterior, se debe buscar mediante la palabra “fifo generator” (Figura 75)

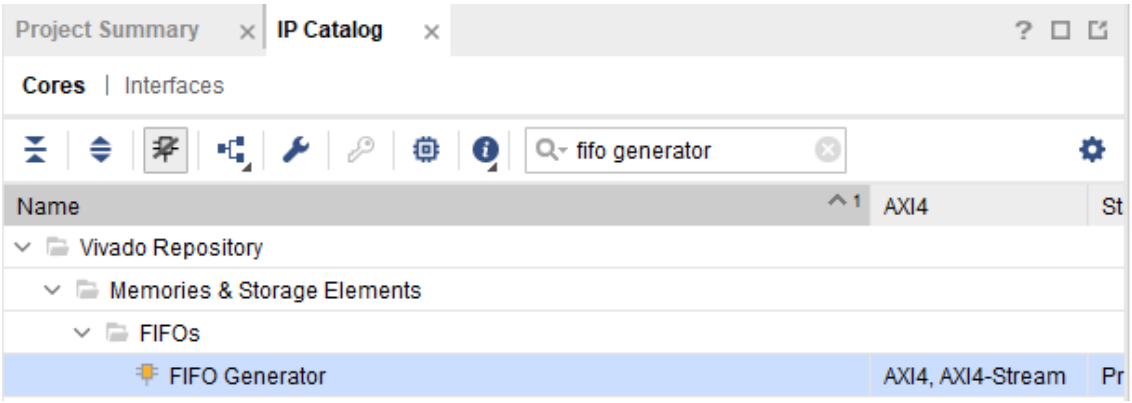


Figura 75: Selección de la FIFO en el catálogo de IPs

El único parámetro que hay que modificar es en la pestaña “Native Ports”, habrá que cambiar el valor Write Width a 16. Automáticamente se modificará el valor Read Width, también a 16 (Figura 76). Para aumentar el tamaño de la FIFO, se podría modificar el parámetro Write Depth, que en este caso se ha dejado a 1024. Esto significa que la FIFO que se creara a continuación tiene un tamaño de 1024 registros de 16 bits cada uno.

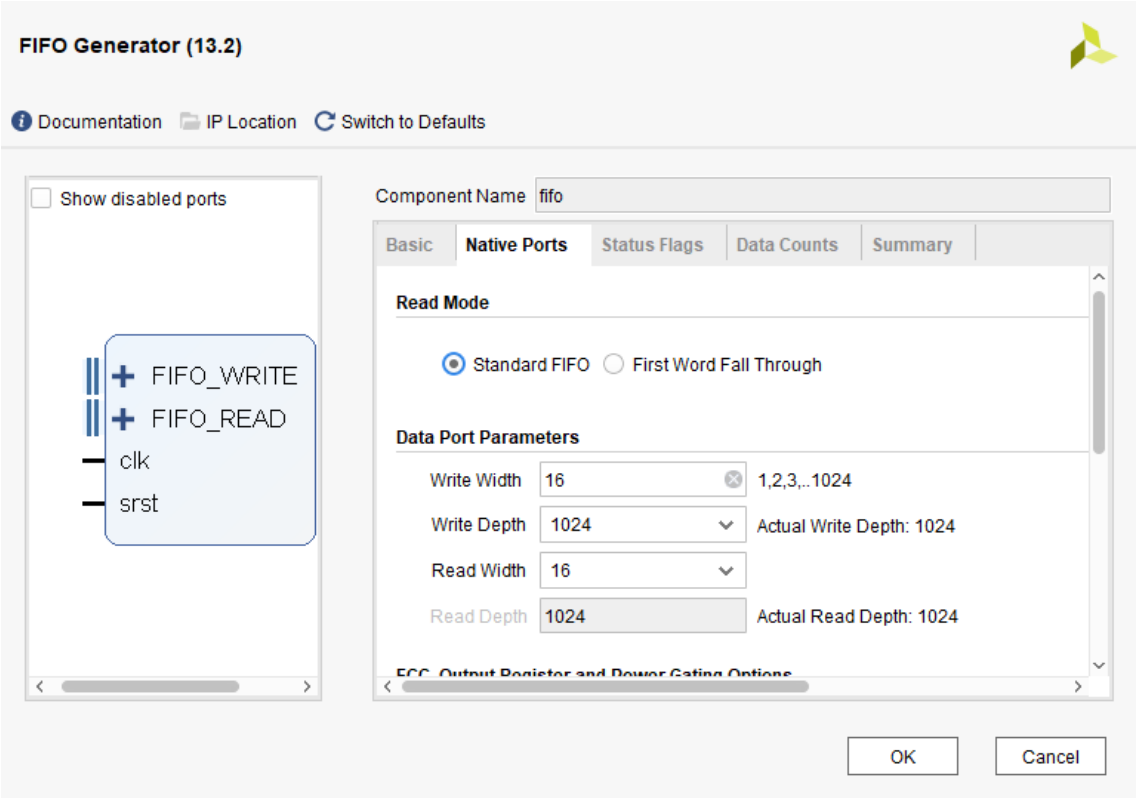


Figura 76: Ajuste de los parámetros de la FIFO

8.2.3.4. Creación del IP de Xillybus

El kit de desarrollo de prueba no sirve para este proyecto, ya que no dispone de una FIFO de 16 bits de ancho y, además, tiene varias funcionalidades que no se utilizan, como la salida de audio de la tarjeta Zedboard. También hay otras funciones de la demo inicial que no están implementadas y que son imprescindibles para este proyecto, como es el caso del ADC.

Para la creación de la IP Core que se necesita, Xilinx dispone de una factoría que lo genera automáticamente, de forma totalmente gratis. Hay que ir a la siguiente dirección web: http://xillybus.com/ipfactory/ y registrarse.

Clicando en añadir un nuevo núcleo, se aparece una pantalla como la de la Figura 77.

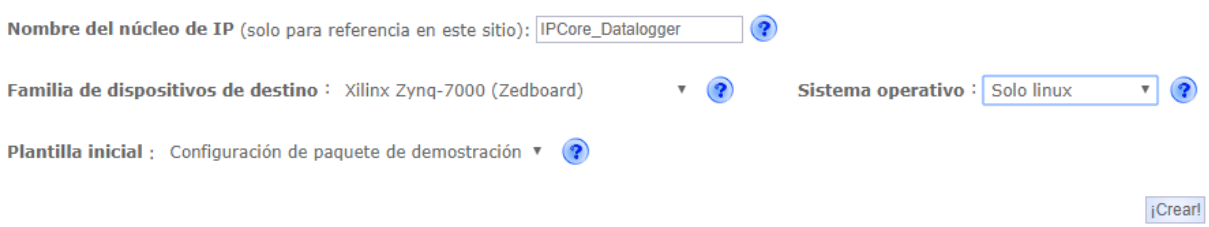


Figura 77:Creación del IP Core de Xillybus

Las opciones a configurar son las siguientes:

- **Nombre del núcleo de IP**: IPCore_Datalogger (el nombre es irrelevante)
- **Familia de dispositivos de destino**: elegir el modelo de FPGA con el que se esté trabajando, en el caso de este proyecto es la Xilinx Zynq-7000 (Zedboard).
- **Sistema operativo**: al elegir la FPGA Xilinx Zynq-7000 solo permite la selección de la opción “Solo Linux”.
- **Plantilla inicial**: lo ideal es seleccionar la opción “Vacía”, pero esta opción ha resultado problemática ya que posteriormente no dejaba añadir nuevos archivos de dispositivo. En dicho caso, se ha seleccionado la opción “Configuración de paquete de demostración” (que es la que trae el kit inicial por defecto) y se han modificado los archivos para crearlos como requiere el proyecto.

El IP Core de Xillybus debe tener dos archivos, como indica la Figura 78.

Archivos del dispositivo

Nombre	Dirección	Ancho de datos	BW esperado	Ajuste automático	Detalles
xillybus_datastream					
	Upstream (FPGA para alojar)	16 bits	102.400 kB / s	Sí	Propósito general
xillybus_controlstream					
	Downstream (anfitrión de FPGA)	8 bits	10.240 kB / s	Sí	Propósito general

Figura 78:Archivos del dispositivo del IP Core creado

Estos archivos son:

- **Xillybus_datastream** (Figura 79): Mediante este archivo se permite el flujo de datos en la dirección de la FPGA al HOST. Los parámetros a ajustar son los siguientes:
  - Nombre: xillybus_datastream
  - Dirección: Upstream (FPGA to Host)
  - Uso: Propósito general
  - Ancho de datos: 16 bits (ya que esta salida se conectará con la FIFO de lectura, cuya anchura es de 16 bits)
  - Ancho de banda esperado: como se ha configura el XADC para que tenga una velocidad de muestreo de 40KSPS, y las muestras son de 16 bits cada una, es decir, 2 bytes/sample. Por lo tanto 40 KSPS * 2 Bytes/sample es igual a 0,08 MB/s. Se pondrá el valor de 0,1 MByte/s.
  - Internos de autoset: activada

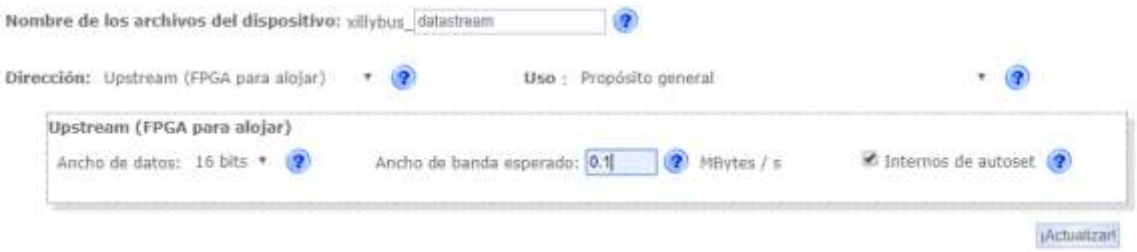


Figura 79:Configuración de xillybus_datastream

- **Xillybus_controlstream** (Figura 80): se trata del archivo que permite el flujo de bytes en la dirección del HOST a la FPGA. Se deberán ajustar los siguientes parámetros, aunque no se haga uso de este archivo en el proyecto, ya que no es necesario ningún parámetro de control del HOST hacia la FPGA.
    - Nombre: xillybus_controlstream
    - Dirección: Downstream (Host to FPGA)
    - Uso: Propósito general
    - Ancho de datos: 8 bits (serán el número de bits que hay que escribir desde Linux para realizar el control)
    - Ancho de banda esperado: 0,01 MByte/s
    - Internos de autoset: activada

Figura 80:Configuración del xillybus_controlstream

El siguiente paso es generar el núcleo con ambos archivos y descargarlo. Para ello se seleccionará clicará sobre la opción "Generar Núcleo". Es posible que tarde unos minutos en aparecer la opción "Descargar". Cuando aparezca, al clicar sobre ella se descargará un .zip con el contenido del IPCore que hay que tratar posteriormente.

Al descomprimir este .zip aparecen varios ficheros que deben remplazar a los que ya hay presentes en el proyecto. Hay que realizar tres cambios:

- Dentro del directorio \xillinux-eval-zedboard-2.0a\vhdl\src hay dos archivos llamados xillybus.v y xillybus_core.v. Estos deben ser eliminados y en su lugar hay que copiar los ficheros con el mismo nombre que aparecen en el zip descargado.
- En el directorio \xillinux-eval-zedboard-2.0a\cores hay un archivo llamado xillybus_core.ngr. Al igual que en el caso anterior, este archivo debe ser remplazado por uno de igual nombre que hay en el zip descargado.

- Editar el top del proyecto, xillybus.vhd. La mayoría de funciones del kit inicial ya no sirven, por tanto, hay que eliminarlos y añadir los dos nuevos dispositivos que se han generado con el nuevo core. En el zip descargado, existe una carpeta llamada "instation templates" con un fichero llamado templates en el que aparece los puertos de entrada y salida del IPCore generado y las conexiones. Se recomienda copiar dicho archivo y añadir las funcionalidades que sean necesarias, como en este caso, la FIFO y el XADC. El top del proyecto xillybus.vhd se encuentra debidamente escrito en el apartado de ANEXOS 1.1. xillydemo.vhd

8.2.3.5. Conexiones de xillybus.vhd

El puerto de entrada y salidas del proyecto total xillybus.vhd es copia del fichero templates, mencionado anteriormente y que se encuentra en el .zip del core descargado, pero hay que añadir el XADC como entradas (Figura 81)

```
port (
clk_100 : IN std_logic;
otg_oc : IN std_logic;
PS_GPIO : INOUT std_logic_vector(55 DOWNTO 0);
GPIO_LED : OUT std_logic_vector(3 DOWNTO 0);
vga4_blue : OUT std_logic_vector(3 DOWNTO 0);
vga4_green : OUT std_logic_vector(3 DOWNTO 0);
vga4_red : OUT std_logic_vector(3 DOWNTO 0);
vga_hsync : OUT std_logic;
vga_vsync : OUT std_logic;
audio_mclk : OUT std_logic;
audio_dac : OUT std_logic;
audio_adc : IN std_logic;
audio_bclk : IN std_logic;
audio_lrclk : IN std_logic;
smb_sclk : OUT std_logic;
smb_sdata : INOUT std_logic;
smbus_addr : OUT std_logic_vector(1 DOWNTO 0);
-- entradas para el XADC
vp_in : IN STD_LOGIC;
vn_in : IN STD_LOGIC;
vauxp0 : IN STD_LOGIC;
vauxn0 : IN STD_LOGIC;
vauxp8 : IN STD_LOGIC;
vauxn8 : IN STD_LOGIC
);
```

Figura 81:Puerto de entradas y salidas del proyecto total xillybus.vhd

Hay que declarar también todos los componentes mencionados que se usarán en dicho proyecto: xillybus, xadc y la fifo.

Estos componentes hay que conectarlos entre ellos de forma que se ajusten al comportamiento que demanda el proyecto:

- El top_xadc se conecta con las entradas del proyecto (vp_in, vn_in, vauxp0, vauxn0, vauxp8 y vauxn8). Se le conecta como entrada clk el reloj del sistema y la salida de los 16 bits de datos se conectan con una señal auxiliar llamada xadc_dout_fifo_din, que posteriormente se conectara con la FIFO (Figura 82)

```
xadc : top_xadc
port map (
clk => bus_clk,
reset => '0',
vp_in => vp_in,
vn_in => vn_in,
vauxp0 => vauxp0,
vauxn0 => vauxn0,
vauxp8 => vauxp8,
vauxn8 => vauxn8,
jtagLMB => open,
dout => xadc_dout_fifo_din,
drdy => xadc_drdy_fifo_wren,
channel => channel_aux
);
```

Figura 82:Conexiones del top_xadc en el proyecto xillydemo.vhd

- La fifo debe estar entre el top_xadc y el xillydemo. Como se trata de una fifo de lectura del host, la entrada din proviene de la señal auxiliar xadc_dout_fifo_din, mientras que la salida de la fifo dout, tiene que ir directamente a xillybus, mediante la señal auxiliar user_r_datastream_data. También hay que destacar que la FIFO se va a resetear mientras que no se esté leyendo (Figura 83).

```
fifo16 : fifo
port map (
clk => bus_clk,
srst => reset_fifo,
din => xadc_dout_fifo_din,
wr_en => xadc_drdy_fifo_wren,
rd_en => user_r_datastream_rden,
dout => user_r_datastream_data,
full => open,
empty => user_r_datastream_empty
);
-- reseteamos la FIFO siempre que no se esté leyendo
reset_fifo <= not(user_r_datastream_open);
-- nunca se alcanza el end of file
user_r_datastream_eof <= '0';
```

Figura 83:Conexiones de la fifo en el proyecto xillydemo.vhd

La FIFO se podrá leer desde el host en el siguiente directorio: /dev/xillybus_datastream

Se pueden ver todas las conexiones en el apartado de ANEXOS 1.1. xillydemo.vhd

8.2.3.6. Archivo de restricciones “constrains”

Este archivo asocia cada nombre de entrada o salida dado en el puerto principal del proyecto a pines de la FPGA Zynq-7000.

En el kit inicial de prueba de Xillydemo viene un archivo cuyas restricciones están asociadas a los pines que utiliza dicha entidad. Como se ha modificado el puerto de entradas y salidas (Figura 81), también ha sido necesario cambiar las restricciones asociadas a dicho puerto.

Se ha añadido como entradas analógicas los canales del XADC (VpVn, Vaux0 y Vaux8). Estas entradas hay que asociarlas a pines de la FPGA. Para saber cuáles son dichos pines se recomienda ir a la Tabla 3.

Estos pines son asociados a las entradas analógicas del proyecto xillydemo.vhd de la siguiente forma:

- vp_in: se asocia con el pin L11
- vn_in: se asocia con el pin M12
- vauxp0: se asocia con el pin F16
- vauxn0: se asocia con el pin E16
- vauxp8: se asocia con el pin D16
- vauxn8: se asocia con el pin D17

Por lo tanto, en el archivo de restricciones xillydemo.xdc se deben añadir las siguientes líneas (Figura 84):

```
# Pins para el XADC
set_property -dict "PACKAGE_PIN L11 IOSTANDARD LVCMOS33" [get_ports "vp_in"]
set_property -dict "PACKAGE_PIN M12 IOSTANDARD LVCMOS33" [get_ports "vn_in"]
set_property -dict "PACKAGE_PIN F16 IOSTANDARD LVCMOS33" [get_ports "vauxp0"]
set_property -dict "PACKAGE_PIN E16 IOSTANDARD LVCMOS33" [get_ports "vauxn0"]
set_property -dict "PACKAGE_PIN D16 IOSTANDARD LVCMOS33" [get_ports "vauxp8"]
set_property -dict "PACKAGE_PIN D17 IOSTANDARD LVCMOS33" [get_ports "vauxn8"]
```

Figura 84:Líneas para añadir las restricciones de las entradas analógicas del XADC.

Además, se ha añadido una línea adicional para desactivar el acceso al XADC mediante JTAG. Aunque esta opción se ha realizado modificando el archivo devicetree.dtb, conviene incluir esta línea (Figura 85) para asegurarse de que esta desactivado.

```
set_property BITSTREAM.GENERAL.JTAG_XADC Disable [current_design]
```

Figura 85:Línea para desactivar el acceso al XADC mediante JTAG.

Por lo tanto, una vez acabado este último paso, se debe general el bitstream, como se ha hecho en con el kit demo inicial (Figura 54). Es posible que la generación del archivo .bit tarde varios minutos.

El código completo de xillydemo.vhd, junto con sus respectivos componentes (xillybus, fifo y xadc) y el archivo de restricciones xillydemo.xdc se pueden consultar en el apartado de ANEXOS. 1. Código VHDL

Por último, al igual que en la prueba del kit inicial, se deben copiar los siguientes cuatro archivos en la tarjeta SD:

- **boot.bin**: sin modificar del kit inicial
- **devicetree.dtb**: modificado según lo explicado en el apartado 8.2.1.3. Modificación del device-tree.
- **uImage**: es el archivo que se ha generado en la prueba de la demo del kit inicial
- **xillydemo.bit**: se trata del archivo modificado en el apartado 8.2.3.6. Archivo de restricciones "constrains" y que se encuentra en el siguiente directorio: \xillinux-eval-zedboard-2.0a\vhdl\vivado\xillydemo.runs\impl_1\xillydemo.bit

8.2.4. Pruebas del diseño final

Las conexiones de los jumpers son las mismas que las descritas en el apartado 8.2.2. Montaje Inicial (Demo), es decir, hay que cortocircuitar los pines JP6, JP10 y JP11.

Tras encender la FPGA y conectar la pantalla a la salida VGA y el teclado y ratón al puerto USB, hay que escribir el comando startx para iniciar el modo gráfico.

Una vez abierto el modo gráfico, abriendo el terminal (Ctrl + Alt + T) para poder configura el teclado de forma correcta en español, hay que escribir el comando:

```
sudo setxbmap –layout 'es,es' –model pc105
```

Es posible que al intentar ejecutar este comando haya problemas de dominio (Figura 86), por lo que hay que editar el hosts del servidor. Para ello hay que ir al directorio /etc/hosts y cambiar la línea:

```
127.0.0.1 localhost
```

Cambiarlo por:

```
127.0.0.1 localhost.localdomain localhost
```

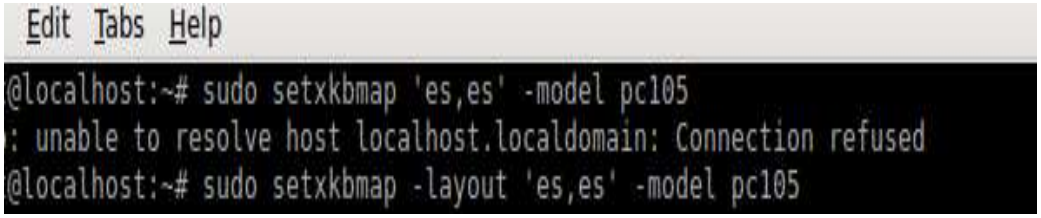


Figura 86:Problemas de dominio del hosts

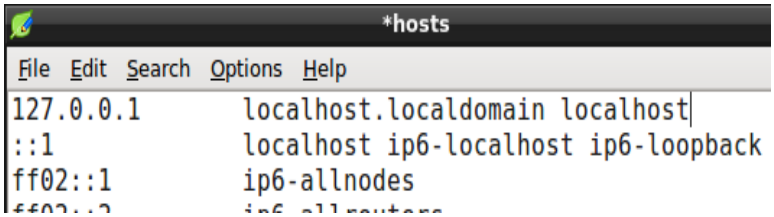


Figura 87: Cambio de dominio del hosts

Tras realizar este cambio (Figura 87) ya no aparecerá más este error.

Para comprobar que se han realizado correctamente los cambios en la SD y que realmente está funcionando correctamente el XADC se escribirá el siguiente comando:

```
hexdump –C /dev/xillybus_datastream
```

El host (Linux) empezara a leer la FIFO y los datos que aparecen son los siguientes (Figura 88)

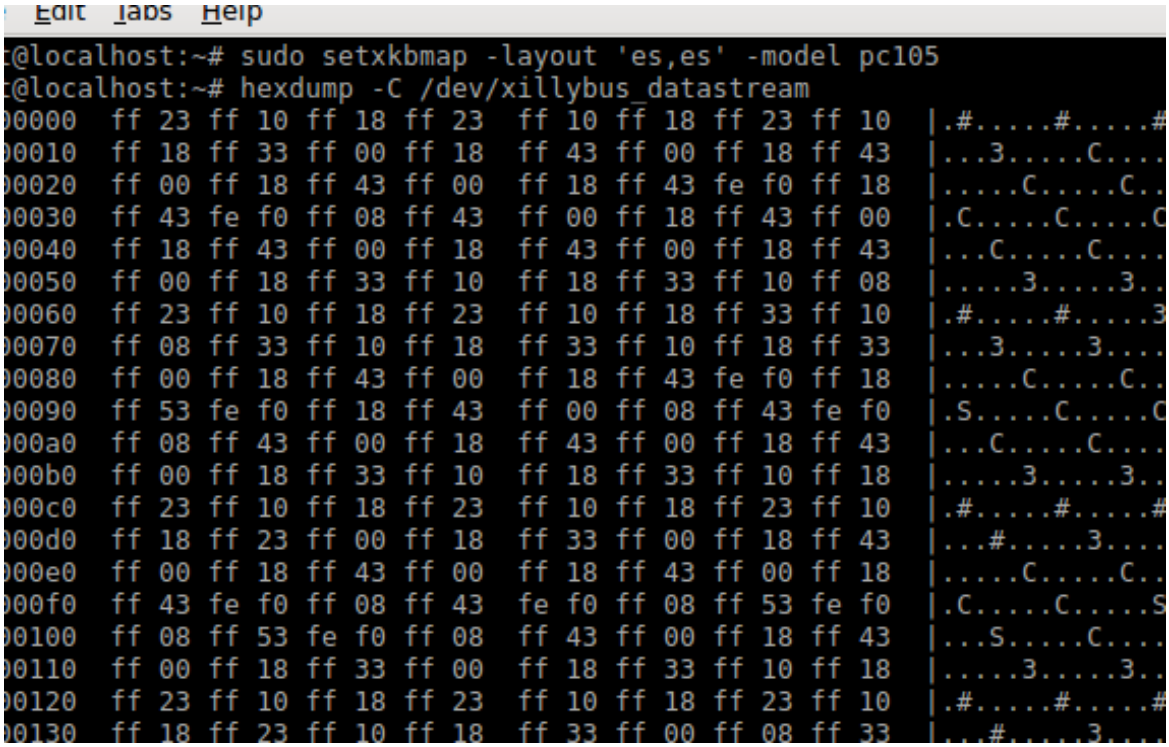


Figura 88: Lectura por parte del Host de la FIFO

Cabe recordar que la FIFO tiene una anchura de 16 bits (2 bytes). Como se observa en la Figura 88, aparecen una pila de bytes que se interpretan de la siguiente manera: cada pareja de bytes corresponde a un valor capturado por el XADC, de forma que los 12 MSB se corresponden al valor del XADC y los 4 LSB se corresponden al canal. Se va a poner de ejemplo la primera línea que se observa en la Figura 88.

f f 2 3	f f 1 0	f f 1 8	f f 2 3	f f 1 0
Valor aleatorio / Canal VpVn	Valor aleatorio / Canal Vaux0	Valor aleatorio / Canal Vaux8	Valor aleatorio / Canal VpVn	Valor aleatorio / Canal Vaux0

Tabla 10: Agrupación de los valores por canales

El valor aleatorio de los datos se debe a que está el XADC está al aire y la señal analógica que está captando se corresponde con el ruido del ambiente. Los últimos 4 bits se corresponden con el canal de captura de la siguiente forma:

- 3: canal dedicado VpVn
- 0: canal auxiliar Vaux0
- 8: canal auxiliar Vaux8

Como se observa en la Figura 88, esta secuencia cambia el valor aleatorio por el valor de entrada al canal, pero mantiene los bits del canal (últimos 4 bits de 16) cambiando de forma periódica entre los valores 3, 0 y 8.

Si quisiéramos tratar estos datos posteriormente, se podría guardar el archivo mediante el comando:

```
cat /dev/xillybus_datastream > ArchivoSalida.xadc
```

Con esta opción podríamos tratar este archivo binario mediante algún interprete, como Python y convertir dichos valores binarios a su valor de tensión real. Para saber cuál es su función de transferencia, se recomienda ir al apartado 8.1.3.6. Función de transferencia.

Esta opción no es interesante para este proyecto, ya que, al tratarse de un *Datalogger*, es conveniente que se realice la conversión y se guarde el valor en tiempo real. El objetivo de este proyecto es realizar una aplicación GUI (Interfaz Gráfica de Usuario) para que el usuario, pueda interactuar con el Host y este a su vez con la FPGA.

Dado que en la tarjeta SD de la ZedBoard está instalado Xillinux, que está basado en Ubuntu 12.04, se ha optado por crear la aplicación desde un ordenador con Linux.

Otra opción es crear la aplicación GUI desde Windows e instalar una aplicación en Xillinux de la FPGA que permita ejecutar programas de Windows en Linux. Un ejemplo de una aplicación con este uso es "wine". Esta opción se descartó porque la tarjeta tiene muy poca capacidad, y tras la instalación de Xillinux, queda muy poca

memoria disponible para instalar los programas (Figura 89). Tan solo se instalarán los programas y librerías imprescindibles.

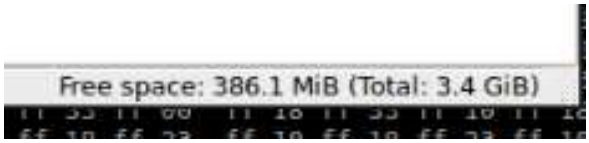


Figura 89:Capacidad de la tarjeta SD tras la instalación de Xillinux.

8.2.5. Instalación de Linux en un ordenador

Existen muchas maneras de poder trabajar con Linux para Escritorio, ya que se trata de un sistema operativo libre y gratuito. Unas opciones son mejores que otras, sobre todo por la velocidad de trabajo que ofrece. Las dos primeras opciones 8.2.5.1. Linux desde máquina virtual y 8.2.5.2. Linux desde arranque con USB han sido descartadas ya que era muy difícil y costoso trabajar mediante estas opciones. Se ha optado por trabajar mediante la tercera opción 8.2.5.4. Partición de disco para Linux ya que era la opción más rápida y fiable.

Se ha elegido el sistema operativo Ubuntu 14.04, ya que se trata del sistema operativo Linux más extendido. Para ello se ha descargado su imagen (archivo con la extensión .iso) desde el siguiente enlace:

https://www.ubuntu.com/download/desktop/thankyou?country=MX&version=14.04.3&architecture=i386

8.2.5.1. Linux desde máquina virtual

Se ha elegido como descarga la máquina virtual Oracle VirtualBox que se puede descargar desde el enlace: https://www.virtualbox.org/wiki/Downloads

Una vez instalada, siguiendo un asistente de instalación, hay que crear la máquina virtual de Ubuntu. Siguiendo el asistente de configuración y seleccionando el archivo de disco de inicio descargado en el apartado 8.2.5 Instalación de Linux en un ordenador se puede disponer de Ubuntu en la máquina virtual (Figura 90).

Esta opción es interesante en el caso de que no se quiera hacer una partición del disco del ordenador. Tiene el inconveniente de que la forma de trabajar es más lenta ya que requiere mucho uso del procesador.

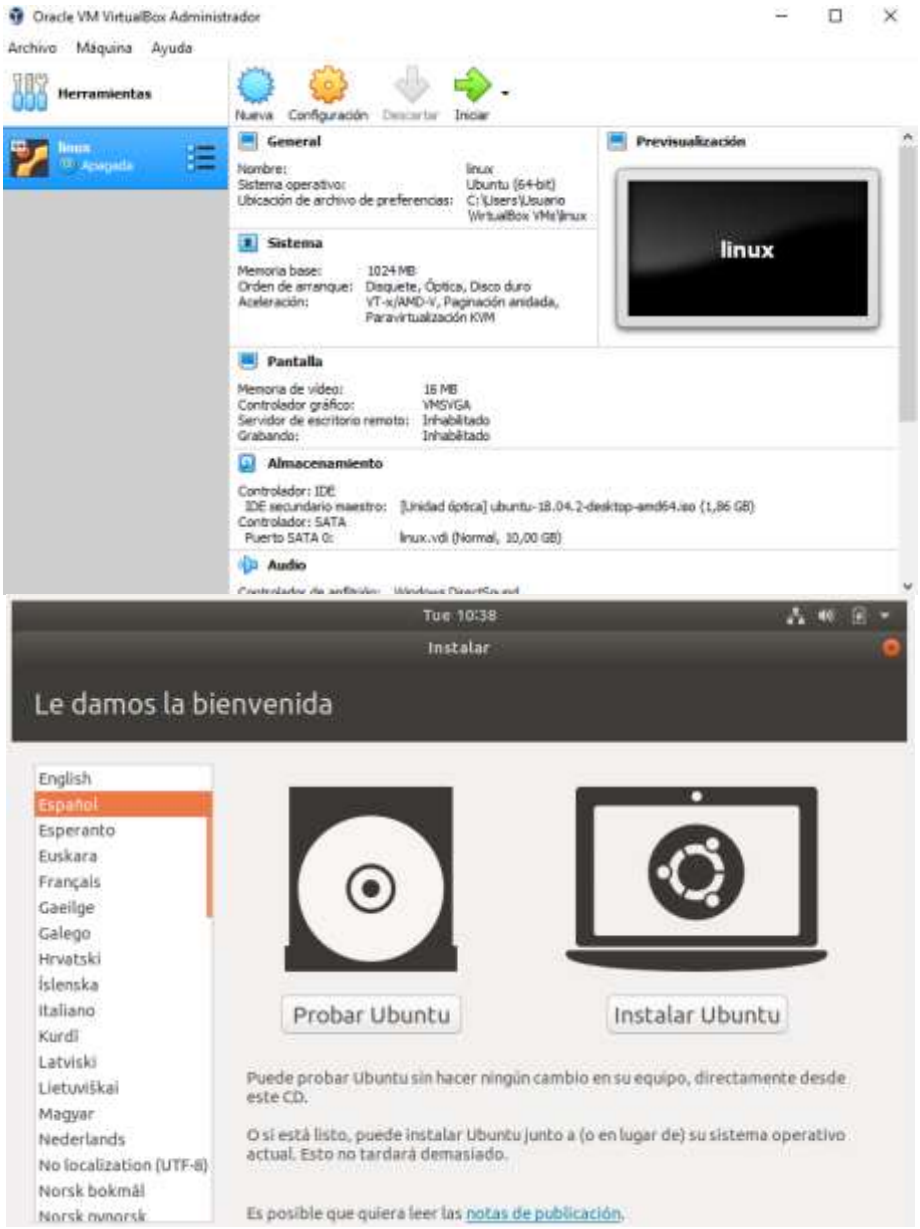


Figura 90: Arriba: máquina virtual VirtualBox. Abajo: visualización de Ubuntu Linux.

8.2.5.2. Arranque desde USB

Para utilizar esta opción se requiere del programa Linux Live USB Creator, disponible en el siguiente enlace: https://www.linuxliveusb.com/download.

Se necesita un USB de mínimo 4 GBytes que no importe formatearlo, ya que antes de proceder a crear la imagen va a ser necesario formatearlo en FAT32. Tras el formateo, se abrirá el programa Linux Live (LiLi)

Las opciones a seleccionar son las que se muestran en la Figura 91.

Figura 91: Opciones de Linux Live Creator para crear una imagen

- **Elija su unidad**: seleccionar el USB formateado en FAT32
- **Elija una fuente**: seleccionar la imagen de Ubuntu descargada
- **Tamaño de persistencia**: poner el máximo disponible, ya que este es el tamaño que usara para almacenar datos que haya dentro del sistema operativo
- **Opciones**: seleccionar todas
- **Creación**: clicar en el rayo para comenzar la creación de la imagen. Este proceso tardara unos minutos. Cuando termine, extraer el USB de forma correcta y ya estará listo para poder usar el sistema operativo Linux desde el USB.

Para poder utilizar el USB como arranque es necesario modificar el BIOS del ordenador.

8.2.5.3. Modificación del BIOS

BIOS corresponde a las siglas Basic Input Output System o lo que es lo mismo Sistema Básico de Entradas y Salidas. Se trata de un estándar que define la interfaz de firmware para ordenadores IBM PC.

El firmware del BIOS es instalado dentro de la computadora personal (PC), y es el primer programa que se ejecuta cuando se enciende la computadora. El BIOS de un

ordenador es un firmware cuyo propósito es activar una máquina desde su encendido y preparar el entorno para cargar un gestor de arranque o un sistema operativo desde un dispositivo de almacenamiento de datos la memoria RAM y en el disco duro. Además, el BIOS provee una capa de abstracción para el hardware, que consiste en una vía para que los programas de aplicaciones y los sistemas operativos interactúen con el teclado, el monitor y otros dispositivos de entrada/salida.

Para modificar el BIOS del ordenador basta con encender el ordenador y rápidamente (antes de que aparezca el símbolo del sistema operativo instalado) pulsar la tecla que se muestra en la siguiente Figura 92 dependiendo de la marca de ordenador. En este caso, al contar con un ordenador Asus, se pulsará la tecla Esc.

Marca de su computadora	Tecla que debe pulsar
Dell	F12
HP	F9
ACER	F12
ASUS	Esc ó F12
LENOVO	FN + F2

Figura 92: Botón de acceso al BIOS según la marca del ordenador.

Cuando se abra, aparecerá un menú de selección como el que se muestra en la Figura 93 Para modificar el acceso del BIOS se debe entrar en la opción Enter Setup.

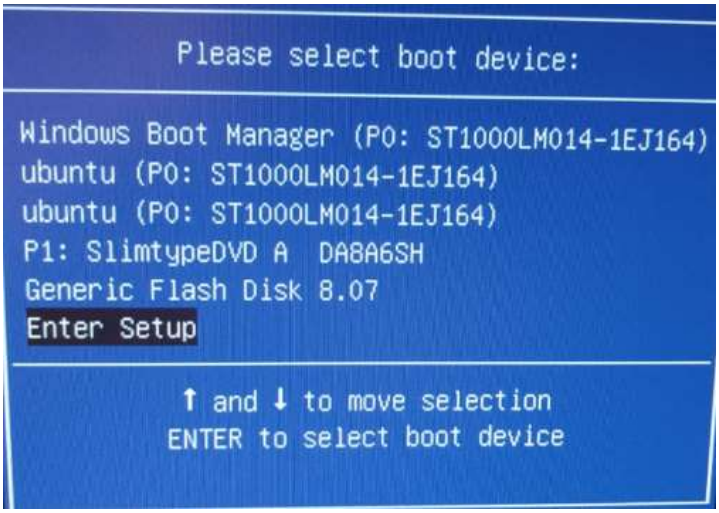


Figura 93:Menú de acceso al BIOS

Para poder permitir que el sistema arranque desde dispositivos externos, es necesario desactivar el modo de arranque seguro. Para ello, se deben colocar las opciones de la siguiente manera, tal y como se indica en la Figura 94:

- En la pestaña Security: Secure Boot Control = Disabled
- En la pestaña Advanced: Legacy USB Support = Enabled

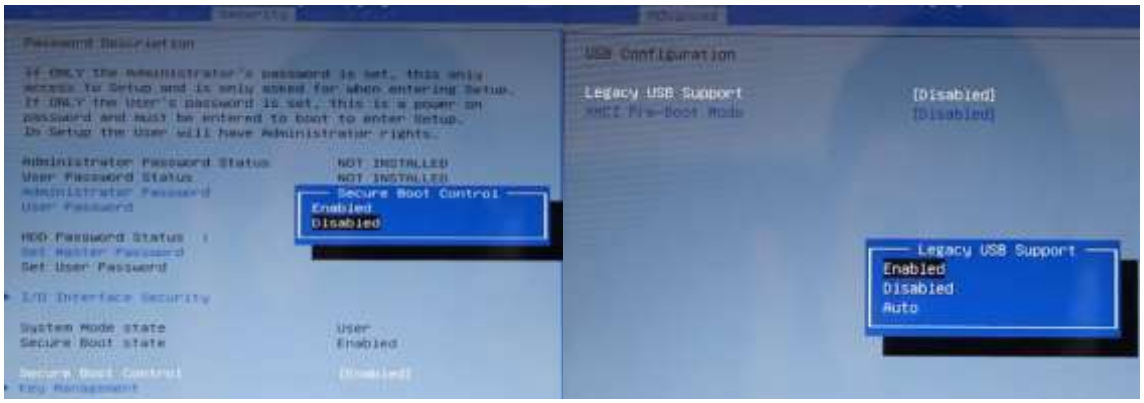


Figura 94:Pestaña Security del BIOS (izquierda) y pestaña Advanced del BIOS (derecha)

Después, hay que cambiar el modo Boot, como se muestra en la Figura 95.

- Launch CSM: Enabled
- Launch PXE OpROM policy: Disabled

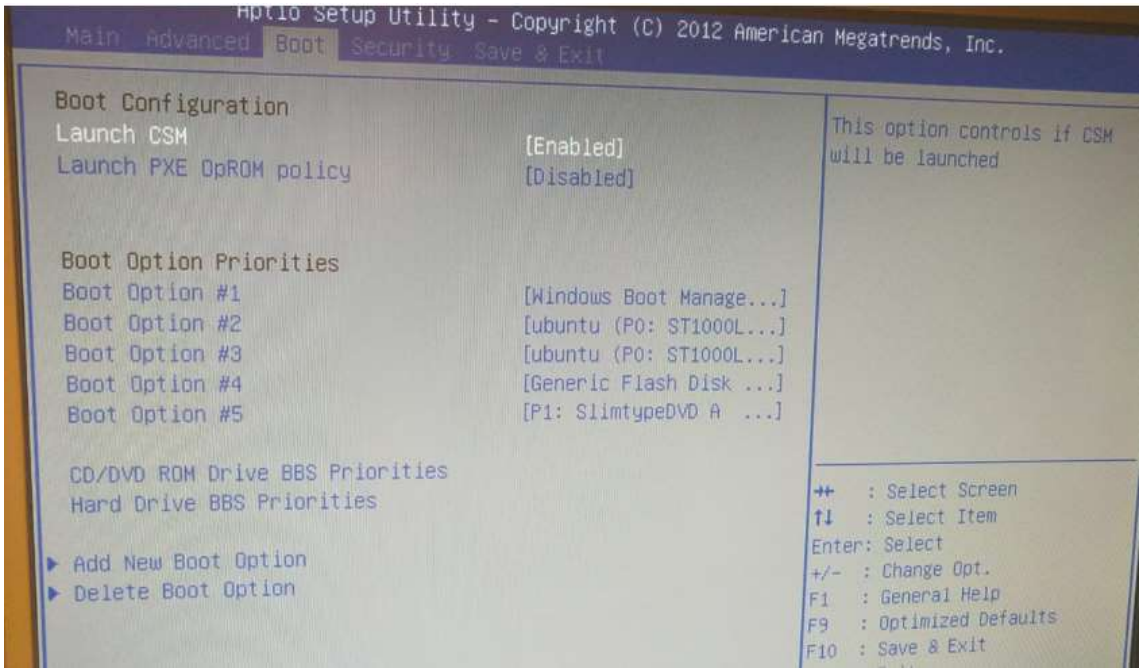


Figura 95:Boot menú del BIOS

También permite configurar el boot que arrancara por defecto y con qué prioridad. En este caso, se tiene por defecto que arranque Windows, ya que es el sistema operativo de trabajo habitual. Si se quiere arrancar desde el USB por defecto, bastaría con cambiar el orden de Boot Options y poner en primer lugar la opción Generic Flash Disk

Si se quiere seguir teniendo por defecto Windows y que solo arranque desde el USB en momento concretos, al arrancar el ordenador, hay que pulsar rápidamente la

tecla Esc y aparecerá la Figura 93. En este caso seleccionar la opción Generic Flash Disk.

Cada vez que se arranque, Ubuntu preguntara si desea instalarlo o probarlo. Para no tener que hacer una partición de disco, si tan solo se pretende tener el sistema operativo en la unidad USB, habría que seleccionar la opción Probar Ubuntu, en lugar de la opción Instalar (Figura 96). Los cambios realizados se guardarán en el USB. Es por eso, que en el apartado 8.2.5.2. Arranque desde USB, se ha seleccionado el máximo tamaño de persistencia posible, ya que estos cambios se almacenarán ahí.

Figura 96:Asistente de instalación. Para guardar cambios en USB: seleccionar Probar Ubuntu. Para hacer partición de disco: seleccionar Instalar Ubuntu.

8.2.5.4. Partición de disco para Linux

Para realizar esta opción, los pasos son los mismos que en el apartado 8.2.5.3. Modificación del BIOS. Es decir, pulsar Esc al iniciar el ordenador, seleccionar la opción Generic Flash Disk y aparecerá la Figura 96. En este caso habrá que seleccionar la opción Instalar Ubuntu y seguir el asistente de instalación.

Lo que se quiere hacer es instalar Ubuntu junto a Windows, en el mismo ordenador, pero sin perder información de Windows. Para ello se debe seleccionar la opción “Instalar Ubuntu junto a Windows 8”, marcada en la Figura 97. Es MUY IMPORTANTE, no seleccionar la opción “Borrar disco e instalar Ubuntu”, ya que esta opción borrara todos los datos de Windows.

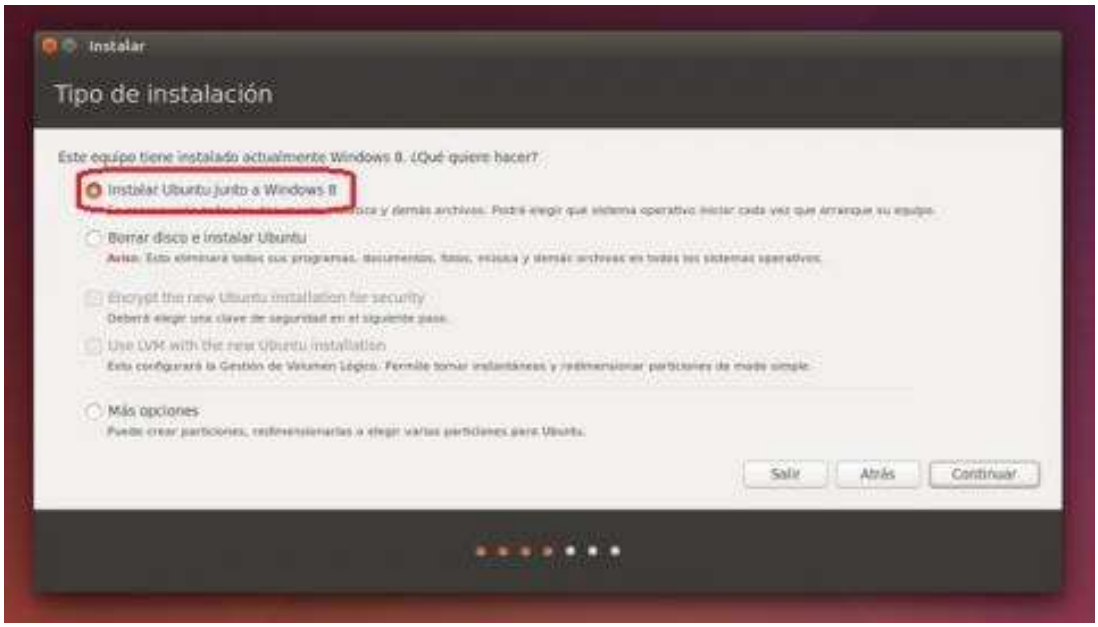


Figura 97:Selección de la opción “Instalar Ubuntu junto a Windows 8”.

Tras darle a continuar, el siguiente paso será decidir cuánto espacio se quiere dejar para cada sistema operativo. Esta decisión va en función del tamaño del disco del ordenador y del uso que se le vaya a dar a cada sistema operativo. En este caso, como Ubuntu solo se utilizará para este proyecto, se le ha dejado 20GBytes, mientras que el resto se han dejado a Windows, que es sistema operativo principal (Figura 98)

Figura 98:Reparto de memoria entre ambos sistemas operativos

Por último, se debe completar el asistente de instalación y esperar varios minutos a que finalice la instalación. Tras la instalación se requiere que se reinicie el ordenador.

Cada vez que se quiera acceder a Ubuntu, hay que abrir el Boot Manager del BIOS, como se ha explicado anteriormente (pulsando la tecla Esc al iniciar el ordenador). Tras aparecer la Figura 98, se seleccionará la opción: ubuntu (P0) y posteriormente la opción que aparece en la Figura 99.

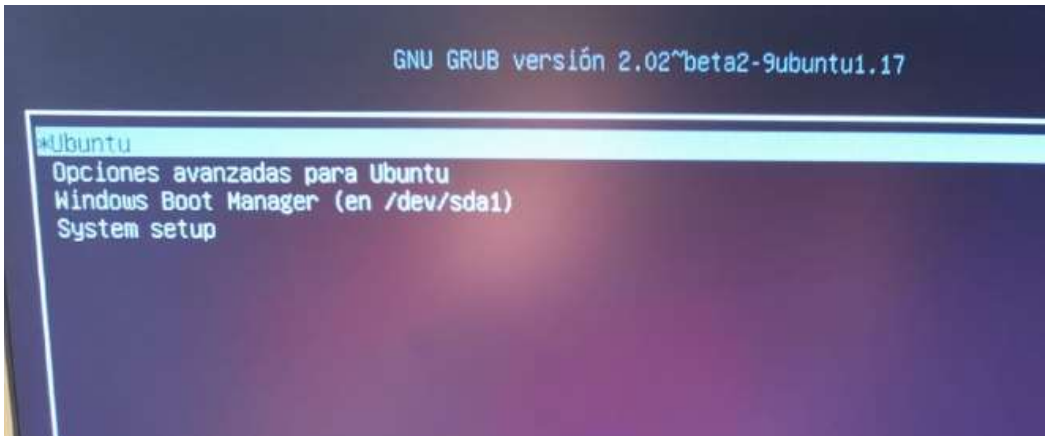


Figura 99:Arranque de Ubuntu

8.2.6. Creación de una aplicación GUI desde Linux

Para la creación de una aplicación que funcione como interfaz de usuario, existen varios programas que permiten su creación. El formato de la tarjeta SD es FAT32, por lo que el máximo espacio disponible para el SO en la tarjeta SD es de 4 GBytes. Debido a que la tarjeta SD cuenta con poco espacio (Figura 8.2.4.4), es necesario usar el que menos espacio consuman sus librerías. Los dos primeros apartados 8.2.6.1 Qt Creator y 8.2.6.2 PyQt son opciones de desarrollo de aplicaciones GUI que se han descartado por falta de espacio. El método utilizado es el descrito en el apartado 8.2.6.3. Tkinter para Python

8.2.6.1. Qt Creator

Qt Creator es un IDE (Integrated Development Environment o Entorno de Desarrollo Integrado) multi-plataforma programado en C++, JavaScript y QML que se utiliza como SDK (Kit de Desarrollo Software) para el desarrollo de aplicaciones con Interfaces Gráficas de Usuario (GUI) con las bibliotecas Qt. Se puede descargar desde el siguiente enlace: https://www.qt.io/download.

Se trata de una herramienta de desarrollo de GUIs de lo más potente que hay en el mercado. Al ser tan potente, utiliza bibliotecas Qt que exceden en mucho la capacidad de la tarjeta SD. Esta opción ha sido descartada, ya que la intención de este proyecto es crear una GUI sencilla que no utilice muchos recursos.

8.2.6.2. PyQt 4

Se trata de una herramienta del mismo fabricante que el anterior apartado 8.2.6.1. Qt Designer, pero esta vez utilizando el lenguaje de programación Python.

Para su instalación en Linux podemos instalar los paquetes de Qt4 utilizando el gestor de paquetes, escribiendo los siguientes comandos (Figura 100):

```
apt-cache search qt4

sudo apt-get install Python-qt4

sudo apt-get install libqt4-designer
```

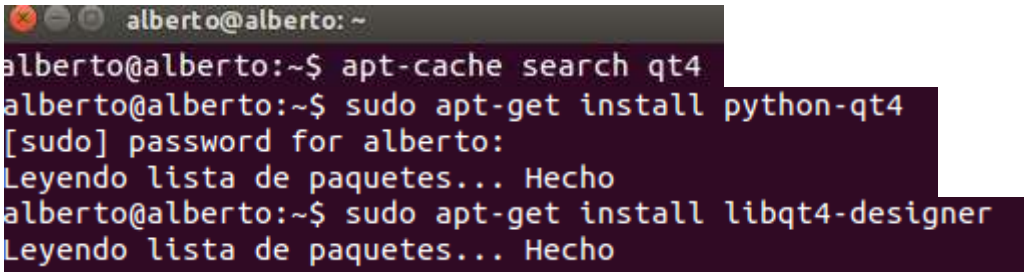


Figura 100:Instalación de las librerías en Ubuntu (en ordenador)

Esto nos instalara el módulo Qt Designer que permite la creación de la aplicación GUI mediante una herramienta de desarrollo visual (Figura 101)

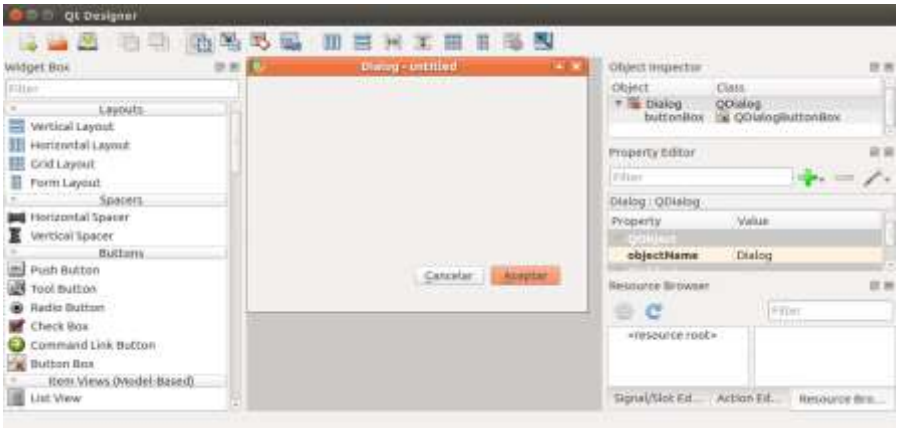


Figura 101:Qt Designer

El archivo que proporciona Qt Designer tiene la extensión .ui, por lo que, si se quiere trabajar en Python, hay que convertir esa extensión a .py. Para ello, la instalación de PyQt contiene un archivo por lotes pyuic4.bat.

En caso de que la instalación de este archivo pyuic4.bat no se haya realizado, habrá que hacerla de forma manual. Para ello hay que ir al siguiente enlace: https://launchpad.net/ubuntu/trusty/amd64/pyqt4-dev-tools/4.10.4+dfsg-1ubuntu1, donde se descargara un archivo con la extensión .deb. Esta extensión es la del formato de paquetes de software de cualquier distribución Linux. Tras ejecutar dicha extensión, ya se podrá ejecutar la instrucción pyuic4 desde el terminal (además de pylupdate4 y pyrcc4). Yendo al directorio donde se encuentra el archivo.ui, se abrirá el terminal y se ejecutará el siguiente comando (Figura 102):

```
pyuic4 nombrearchivo.ui -o nombrearchivo.py
```

```
alberto@alberto:~/Escritorio/disenoQTDesigner$ pyuic4 nombrearchivo.ui -o nombre
archivo.py
alberto@alberto:~/Escritorio/disenoQTDesigner$
```

Figura 102: Comando para cambiar la extensión .ui a .py

En el directorio donde se encuentra el fichero nombrearchivo.ui, aparecerá otro con el nombre “nombrearchivo.py” (Figura 103). Este último ya lo podremos modificar escribiendo en Python.

Figura 103:Creación del archivo con extensión .py

Este apartado se trata de un ejemplo que no se ha llevado a cabo, ya que la tarjeta SD es incapaz de importar las librerías y bibliotecas de PyQt4, por falta de espacio.

En caso de utilizar una tarjeta más con más capacidad y tener conocimientos de programación orientada a objetos, se recomienda utilizar este método de creación de GUIs por su sencillez y la amplitud de aplicaciones que se pueden crear.

8.2.6.3. Tkinter para Python

Lo primero de todo es tener un entorno de desarrollo integrado (IDE) para Python, lo que permitirá editar el código del programa y a su vez, poder ejecutarlo e incluso ver la ventana grafica que se está creando. Existen varios IDEs para Python:

- PyDev
- PyCharm
- Vim
- Spyder
- Django
- Wing IDE

De todas estas opciones, se ha elegido instalar el IDE Spyder, debido a su sencillez de ejecución y a que ofrece una combinación de edición, análisis, depuración y creación de perfiles muy avanzada. Además, Spyder también se puede utilizar como una

biblioteca de extensión de PyQt, lo que permite integrar componentes como la consola interactiva en su propio software.

Se utilizará el lenguaje Python 2 en lugar de Python 3. Esto es importante, ya que la mayoría de instalaciones hay que especificar qué tipo de Python se requiere, como, por ejemplo, para la instalación del IDE Spyder, en la que hay que ejecutar el siguiente comando:

- Si se utilizar Python 2: `sudo apt-get install Spyder` (Figura 104)
- Si se utiliza Python 3: `sudo apt-get install spyder3`

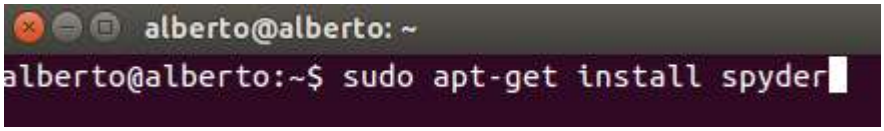


Figura 104:Comando de instalación del IDE Spyder

Para la interfaz gráfica de usuario (GUI) existen varias librerías que se pueden utilizar como componente visual. Estas librerías son:

- wxPython
- PyQt
- Pyside
- PyGTK
- Tkinter

Se ha elegido esta última opción Tkinter, debido a su reducido consumo de recursos de memoria y a su sencillez. Además, se recomienda instalar librerías y dependencias que se utilizaran posteriormente en la escritura del código de la aplicación. Para instalar estas librerías hay que escribir en el terminal los comandos descritos en la Tabla 11.

Nombre de la librería	Comando de instalación
Tkinter	Sudo apt-get install python-tk
Setuptools	Sudo apt-get install python-setuptools
Suds	Sudo apt-get install suds
Matplotlib	Sudo apt-get install python-matplotlib
Numpy	Sudo apt-get install python-numpy
Scipy	Sudo apt-get install python-scipy
Imagetk	Sudo apt-get install python imaging python-imaging-tk

Patsy	Sudo easy_install patsy
Pandas	Sudo easy_install pandas
Fastcluster	Sudo easy_install fatcluster
Statistics	Sudo easy_install statistics

Tabla 11: Librerías a instalar de Python

Para instalar paquetes de Python, es necesario tener su instalador “pip”. PIP significa "Paquetes de instalación PIP" y se trata de una utilidad de línea de comandos que permite instalar, reinstalar o desinstalar paquetes PyPI con un comando simple y directo: "pip".

Para instalar PIP, es necesario comprobar que Python está correctamente instalado en el sistema. Se puede comprobar escribiendo en el terminal:

- Si se cuenta con python 2: `python -- version` (caso de este proyecto). Se muestra un ejemplo en la Figura 105.
- Si se cuenta con python 3: `python3 -- version`

Para instalar PIP, se debe escribir en el terminal el comando:

- Si se cuenta con python 2: `sudo apt-get install python-pip` (Figura 105)
- Si se cuenta con python 3: `sudo apt-get install python3-pip`

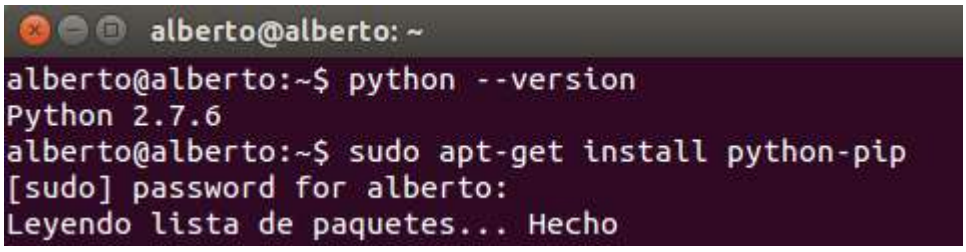


Figura 105:Version de Python e instalación de PIP

Todas las librerías que se instalen en el ordenador con Linux, también deben ser instaladas en la FPGA Zynq-7000, ya que sino la aplicación no se podrá leer desde la tarjeta. Es decir, los comandos de la Tabla 11, deben ser también escritos en la Xillinux de la FPGA.

8.2.7. Código de la aplicación GUI para un *Datalogger*

La interfaz de usuario debe permitir al usuario interactuar directamente con el software, sin la necesidad de teclear ningún código. Para ello se ha creado dicha aplicación. Para escribir este código, se tiene que crear un fichero con la extensión .py.

Esto se puede realizar creando un nuevo archivo de texto y al cambiar el nombre, cambiar el “.txt” por “.py”. De esta manera, ya se puede abrir este fichero.py en el programa Spyder descargado previamente en el apartado 8.2.6.3 Tkinter para Python.

Lo siguiente es importar todas las librerías de la Tabla 11, como se indica en la Figura 106.

```
import matplotlib as mpl
import numpy as np
import sys
from Tkinter import *
import Tkinter as tk
import time
import binascii
import os
import sys
sys.setrecursionlimit(1000000)
from datetime import date, datetime
import matplotlib.backends.tkagg as tkagg
from matplotlib.backends.backend_agg import FigureCanvasAgg
import csv
import statistics as stats
import threading
import Queue
import tkMessageBox
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.base import MIMEBase
from email import encoders
```

Figura 106:Librerías importadas en Python

Para que el código en Python pueda ser lanzado como un ejecutable, es necesario escribir en la primera línea lo siguiente:

```
#! /usr/bin/env python
```

Lo principal es crear una ventana donde se puedan alojar los botones, gráficos, datos, etc, para que el usuario pueda interactuar con el software. En esta ventana existen cuatro botones principales y una entrada de texto de configuración:

- START: empieza a contar el tiempo para la muestra de datos
- STOP: detiene el tiempo de muestreo
- CONFIGURACION: abre otra ventana que permite ajustar los tiempos en que el usuario desea que se muestreen datos.
- Mostrar/Ocultar Datos: abre un desplegable donde se muestran los cuatro últimos valores capturados de cada canal
- Email: permite establecer el correo electrónico al cual se enviarán los emails de copia de seguridad. También permite ajustar el tiempo cada cuanto el usuario desea que se envíen las copias de seguridad. Se ha creado un correo electrónico nuevo para que actué como remitente del envió de las copias de seguridad.

usuario: TFGAlbertoOtano@gmail.com

contraseña: tfgalberto

Dentro de la configuración de Gmail, hay que permitir que aplicaciones externas puedan acceder al correo (Figura 107)

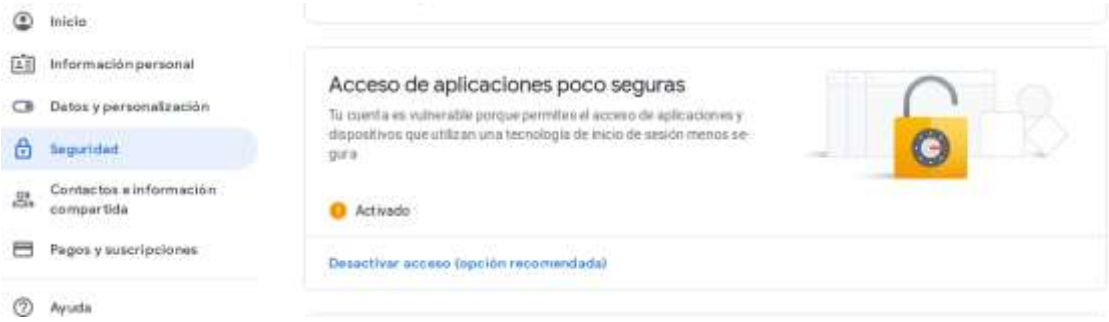


Figura 107:Acceso a aplicaciones poco seguras

El código de ejecución de la aplicación se encuentra en el apartado de ANEXOS 2. Código Python. Se procederá a resumir su funcionamiento:

Se trata de una aplicación GUI con una imagen fondo de interfaz de usuario. Para la edición y modificación de la imagen se ha necesitado el programa GIMP. Se puede descargar desde el siguiente enlace: https://gimp.uptodown.com/ubuntu o desde la terminal con los siguientes comandos:

```
sudo add-apt-repository ppa:otto-kesselgulasch/gimp

sudo apt update

sudo apt install gimp
```

Dentro de la ventana principal existen los cinco botones mencionados anteriormente. Al pulsar el botón START se crea un bucle infinito “while” en el que compara la hora asignada en la pestaña configuración con la hora actual. Para obtener la hora actual llama a la función del sistema “datetime.now()”. Dentro del bucle “while” se comparan tres canales (0, 1 y 2). En caso de que la hora del sistema sea igual a la programada, llama a la función CaptarDato(), que obtiene el valor del dato en el conversor y GuardarDato(canal, dato) al que le pasa el canal del que se tiene que obtener la información y el dato captado en la función anterior CaptarDato(). La función GuardarDato() escribe en el fichero CSV correspondiente. El uso y llamada a funciones se explica detalladamente en la siguiente Figura 108.

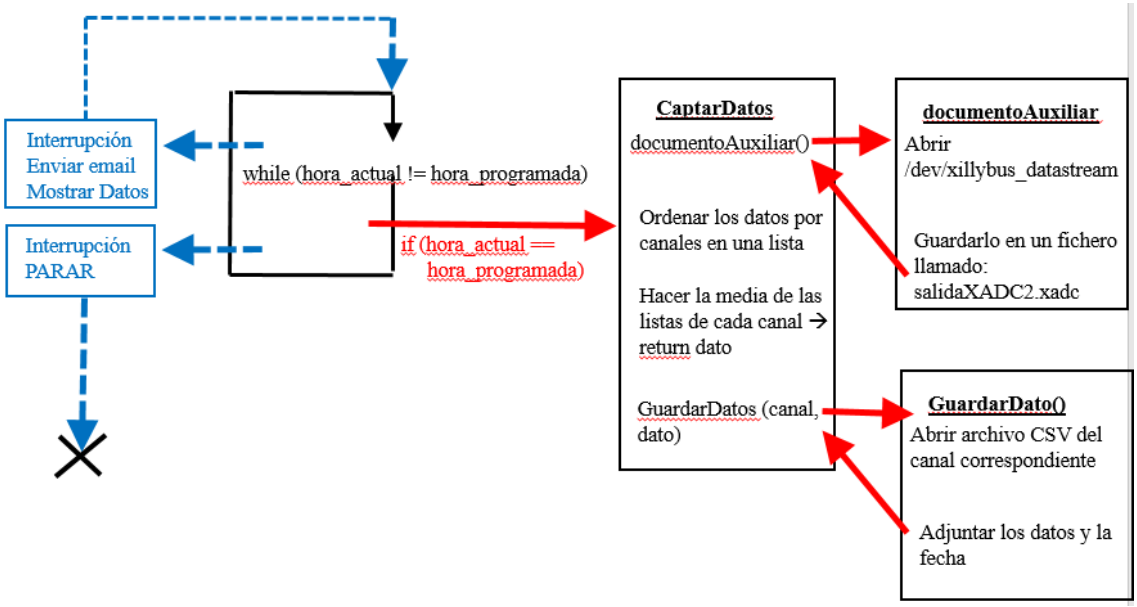


Figura 108: Llamamiento a funciones del código principal.

Una vez pulsado START se mete en un bucle infinito. Para salir del hay que generar que se puedan producir interrupciones por botones. Las interrupciones permiten que, aunque este dentro del bucle, se pueda modificar la interfaz de usuario y este pueda interactuar con los botones, etc. De lo contrario, se bloquearía la aplicación cuando se pulse START. Esto se consigue mediante las siguientes líneas de código:

```
def STOP():
    global PARAR
    PARAR=1
    cola.put((selPARAR.set(1), ("Exito", "La persiana se ha cerrado"), {}))

def MOSTRAR():
    global MOSTRAR
    ventana.geometry("600x850")
    bMOSTRAR.place_forget()
    bOCULTAR.place(x=450,y=230)
    MOSTRAR=1
    cola.put((selMOSTRAR.set(1), ("Exito", "La persiana se ha cerrado"), {}))
def OCULTAR():
    global OCULTAR
    ventana.geometry("600x320")
    bOCULTAR.place_forget()
    bMOSTRAR.place(x=450,y=230)
    MOSTRAR=0
    cola.put((selMOSTRAR.set(0), ("Exito", "La persiana se ha cerrado"), {}))

def activarEnviar():
    cola.put((selENVIAR.set(1), ("Exito", "La persiana se ha cerrado"), {}))
    enviarEmail()
def tkloop():
    try:
        while True:
            funcion, args, kwargs  = cola.get_nowait()
            funcion(*args, **kwargs)
            time.sleep(1)
    except:
        pass

    ventana.after(100, tkloop)

tkloop()
```

Figura 109:Creación de interrupciones

Para el envío de email, se ha creado una función llamada “enviarEmail()”, que crea un servidor con la función smtplib y que le adjunta las rutas de los archivos que se pretenden enviar (Figura 110)

```
def enviarEmail():
    x=datetime.now()
    print(x)
    fromaddr = 'TFGAlbertoOtano@gmail.com'
    toaddrs  = 'albertootano1997@gmail.com'
    asunto = ('Datos hasta la fecha: '+ str (x))
    cuerpo = ('Fecha de envio del mensaje:' + str (x))
    mensaje = MIMEMultipart()
    # Establecemos los atributos del mensaje
    mensaje['From'] = fromaddr
    mensaje['To'] = ", ".join(toaddrs)
    mensaje['Subject'] = asunto
    # Creamos el objeto mensaje
    ruta_adjunto0 = '/home/alberto/Escritorio/DATALOGGER2_tarjeta/CSV/canal0.csv'
    nombre_adjunto0 = 'canalTemperatura.csv'
    ruta_adjunto1 = '/home/alberto/Escritorio/DATALOGGER2_tarjeta/CSV/canal1.csv'
    nombre_adjunto1 = 'canalHumedad.csv'
    ruta_adjunto2 = '/home/alberto/Escritorio/DATALOGGER2_tarjeta/CSV/canal2.csv'
    nombre_adjunto2 = 'canalLluvia.csv'
    # Agregamos el cuerpo del mensaje como objeto MIME de tipo texto
    cuerpo = 'Este es el contenido del mensaje'
    mensaje.attach(MIMEText(cuerpo, 'plain'))
    # Abrimos el archivo que vamos a adjuntar
    archivo_adjunto0 = open(ruta_adjunto0, 'rb')
    archivo_adjunto1 = open(ruta_adjunto1, 'rb')
    archivo_adjunto2 = open(ruta_adjunto2, 'rb')
    # Creamos un objeto MIME base
    adjunto0_MIME = MIMEBase('application', 'octet-stream')
    adjunto1_MIME = MIMEBase('application', 'octet-stream')
    adjunto2_MIME = MIMEBase('application', 'octet-stream')
    # Y le cargamos el archivo adjunto
    adjunto0_MIME.set_payload((archivo_adjunto0).read())
    adjunto1_MIME.set_payload((archivo_adjunto1).read())
    adjunto2_MIME.set_payload((archivo_adjunto2).read())
    # Codificamos el objeto en BASE64
    encoders.encode_base64(adjunto0_MIME)
    encoders.encode_base64(adjunto1_MIME)
    encoders.encode_base64(adjunto2_MIME)
    # Agregamos una cabecera al objeto
    adjunto0_MIME.add_header('Content-Disposition', "attachment; filename= %s" % nombre_adjunto0)
    adjunto1_MIME.add_header('Content-Disposition', "attachment; filename= %s" % nombre_adjunto1)
    adjunto2_MIME.add_header('Content-Disposition', "attachment; filename= %s" % nombre_adjunto2)
    # Y finalmente lo agregamos al mensaje
    mensaje.attach(adjunto0_MIME)
    mensaje.attach(adjunto1_MIME)
    mensaje.attach(adjunto2_MIME)
    mensaje2 = mensaje.as_string()
    username = 'TFGAlbertoOtano@gmail.com'
    password = 'tfgalberto'
    #creacion del servidor
    server = smtplib.SMTP('smtp.gmail.com:587')
    server.starttls()
    server.login(username,password)
    server.sendmail(fromaddr, toaddrs,mensaje2)
    server.quit()
```

Figura 110: Función de enviar emails

8.3. Montaje de circuito de simulación

8.3.1. Desarrollo teórico

El prototipo de *Datalogger* para una estación meteorológica no contará con sensores reales de temperatura, humedad y nivel para lluvia, sino que se sustituirán por potenciómetros. Para poder realizar la simulación se han de implementar circuitos analógicos, mediante amplificadores operacionales. Se pretende transformar la tensión

de salida del potenciómetro, cuya tensión máxima será de 5V y la mínima de 0V a la tensión diferencial de entrada al conversor XADC, cuyos valores son 0,5V de máxima y 0,5V de mínima. Por lo tanto, se pretende realizar la función de transferencia de la Figura 111.

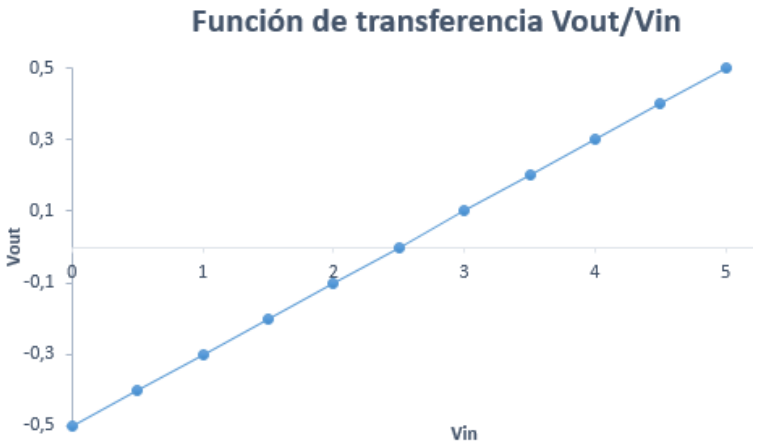


Figura 111: Función de transferencia Vout/Vin

Se han de realizar tres circuitos iguales entre ellos, uno para cada canal. Para ello se pretende hacer uso de dos amplificadores operaciones que funcionen como diferencial o restador e inversor. El objetivo es obtener la función de transferencia de la Figura 111, que es la siguiente:

$$Vout = -0{,}5 + 0{,}2\,Vpot$$

Siendo Vout la tensión de salida del circuito y entrada al XADC y Vpot la tensión de salida del potenciómetro (entre 0 y 5 V)

Se empezará por un amplificador operacional que funcione como restador, cuyo circuito se muestra en la Figura 112 y la ecuación de salida en función de la entrada es la siguiente:

$$Vout = Vt * \left(\frac{R4}{R3 + R4}\right) * \left(1 + \frac{R2}{R1}\right) - Vdc * \frac{R2}{R1}$$

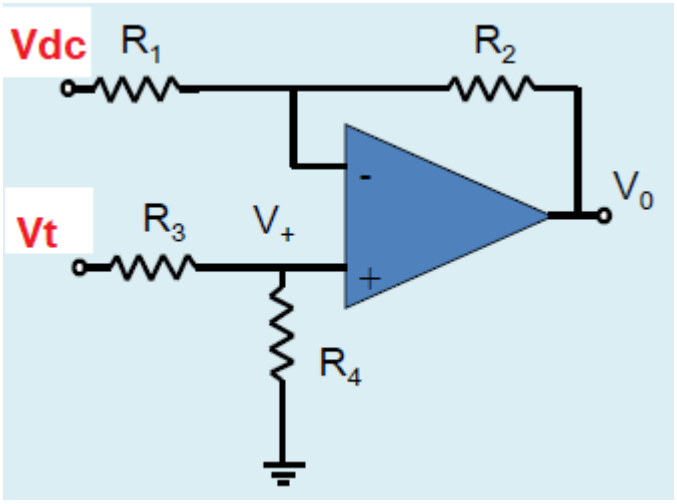


Figura 112:Circuito restador con amplificador operacional

Igualando términos entre la función de transferencia deseada y la que nos proporciona el amplificador operacional obtenemos que:

- $0{,}5 = \frac{R2}{R1}$
- $0{,}2 = \left(\frac{R4}{R3+R4}\right) * \left(1 + \frac{R2}{R1}\right) * Vt$

Suponiendo Vdc como la tensión de alimentación de todo el circuito, es decir, 5 V, se ha fijado R2 = 1K y R4=1K y se ha obtenido que R1 = 10K y R3 = 4,5K. Dado que 4,5K no es un valor de una resistencia comercial, se han sustituido por una de 4,7K, ya que la diferencia de tensión va a ser mínima (Tabla 12).

R1	R2	R3	R4
10K	1K	4,7K	1K

Tabla 12:Tabla de resistencias a colocar en el circuito

8.3.2. Simulación en PSIM

Para comprobar que el comportamiento del circuito anterior es el deseado, se cuenta con el software PSIM, que permite simular circuitos eléctricos y electrónicos de forma muy sencilla. Tras dibujar el circuito en su interfaz gráfica (Figura 113), se ha obtenido el comportamiento deseado en el tiempo (Figura 114).

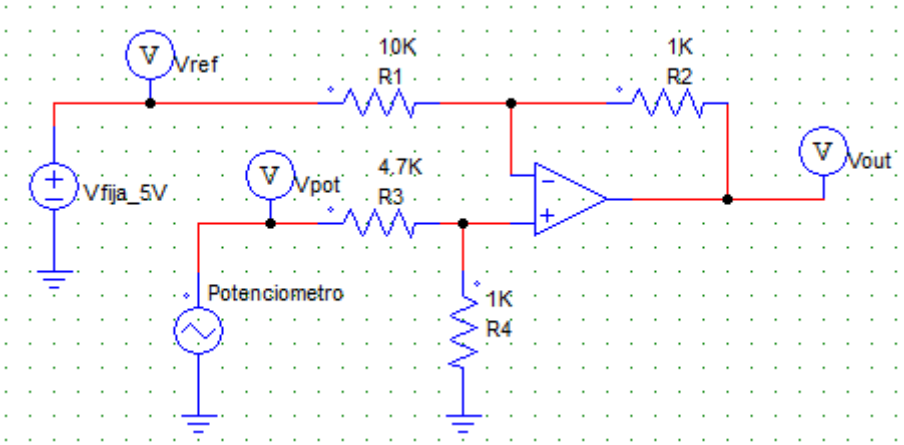


Figura 113: Dibujo del circuito en PSIM

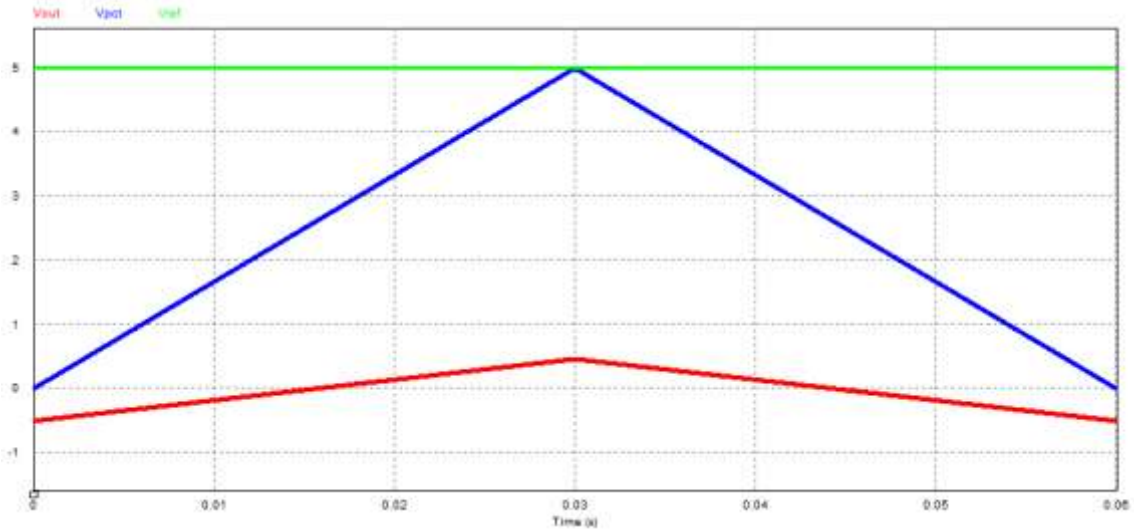


Figura 114: Resultados de la simulación

En la Figura 114, la señal verde es la tensión de referencia de 5 V, la señal azul son los diferentes rangos de valor del potenciómetro (de 0 a 5 V) y la tensión roja es la tensión de salida del circuito (de -0,5V a 0,5V).

8.4. Otras funciones

Como propuestas de mejora de la aplicación, se ha pensado en la realización de otras funciones como:

- El envío de un SMS a un teléfono móvil indicando que se ha sobrepasado alguno de los umbrales máximo o mínimo establecidos por el usuario. Esta función ya la realiza el sistema mediante el envío de un email, pero para hacerlo de forma más segura, también se podría implementar este servicio.
- Enviar un SMS que informe al usuario de que no hay red, por lo tanto, no se va a poder producir el salvaguardado de los datos tomados.

- Mandar un correo electrónico al usuario informándole que ha dejado de funcionar el servicio de SMS, por lo que no va a poder recibir las alarmas por desbordamiento de umbrales.
- Enviar un correo electrónico, SMS o ambas, informando al usuario de la capacidad restante del dispositivo USB donde se almacene la información. Se avisaría en los casos en que el dispositivo de almacenamiento tenga tan solo un 25 % de capacidad restante, un 10 % y un 5 %.

Para el envío de SMS es necesario un módulo GSM con conexión USB, con el que no se ha contado durante el proyecto, por lo que no se ha podido realizar esta función. Se puede disponer de un módulo de este estilo en el siguiente enlace: https://www.amazon.es/SIM900-M%C3%B3dulo-Placa-desarrollo-Arduino/dp/B01FK32ODM [Último acceso: 24/06/2019].

El objetivo es enviar comandos AT mediante el puerto USB para que sea interpretado por el módulo GSM y haga la función específica del comando. El conjunto de comandos con sus respectivas funciones se encuentra en la siguiente Tabla 13:

AT	Comprueba estado del módulo.
AT+CPIN="XXXX"	Introducir el PIN de la SIM. Cambiar XXXX por el PIN
AT+CREG?	Comprueba la conexión a la red.
ATDXXXXXXXX;	Realiza una llamada. Sustituir XXXXXXXXX por el nº al que queramos llamar.
ATA	Descuelga una llamada.
ATH	Finaliza la llamada.
AT+CMGF=1	Configura el modo texto para enviar o recibir mensajes. Devuelve ">" como inductor.
AT+CMGS="XXXXXXXXX"	Nº al que vamos a enviar el mensaje.
AT+CLIP=1	Activamos la identificación de llamada.
AT+CNMI=2,2,0,0,0	Configuramos el módulo para que muestre los SMS por el puerto serie.
AT+CGATT=1	Conectamos a la red GPRS.
AT+CSTT="APN","usuario","contraseña"	Definimos APN, usuario y contraseña
AT+CIICR	Activamos el perfil de datos inalámbrico
AT+CIFSR	Obtenemos nuestra IP
AT+CIPSTART="TCP","direccionIP","puerto"	Indicamos el tipo de conexión, dirección IP y puerto al que realizamos la conexión
AT+CIPSEND	Preparamos el envío de datos. Devuelve ">" como inductor.
AT+CIPCLOSE	Cerramos la conexión.
AT+CIPSHUT	Cierra el contexto PDP del GPRS
AT+CGPSPWR=1	Activar el GPS.
AT+CGPSSTATUS?	Comprueba que el GPS ha encontrado la red.
AT+CGPSINF=0	Obtiene los datos del GPS. Modo,Latitud,Longitud,Altitud,HorarioUTC,Tiempo de respuesta,Número de satélites,Velocidad,Curso
AT+CGPSOUT=32	para obtener los datos del GPS usando la especificación de la NMEA. Horario UTC,Estado,Latitud,Longitud,Velocidad en nudos,Ángulo de derrota en grados,Fecha,Variación magnética,Datos del Cheksum
AT+CGPSPWR=0	Cerrar el GPS.

Tabla 13: Comando AT para los modulos GSM/GPRS y GPS SIM900 y SIM808

Por lo tanto, el código comentado para enviar SMS es el que se observa en la Figura 115.

```
#!/usr/bin/env python

import serial
# para enviar el Ctrl-Z
from curses import ascii

sSerie = serial.Serial('/dev/ttyUSB2', 9600)
#sSerie.open()
try:
    # Enviamos un reset al modem
    sSerie.write("ATZ\r\n")
    # Le ponemos en modo para SMS
    sSerie.write("AT+CMGF=1\r\n")
    # Le pasamos el numero al que vamos ha mandar el SMS
    sSerie.write("AT+CMGS=\"123456789\"\r\n");
    # Texto del mensaje terminado en Ctrl+Z
    sSerie.write("Esto es un mensaje" + ascii.ctrl('z'))
    # Leemos la informacion devuelta
    print sSerie.readline()
    # Leemos la informacion devuelta
    print sSerie.readline()
    # Leemos la informacion devuelta
    print sSerie.readline()

except ValueError:
    print "Oops! se ha producido un error ..."

#sSerie.close()
```

Figura 115: Código en Python comentado para el envío de SMS

La información se enviará por el puerto serie, cuyas direcciones son las adjuntadas en la siguiente Tabla 14.

Puerto	Dirección Entrada	Dirección Salida
COM 1	dev/ttyS0	dev/cua0
COM 2	dev/ttyS0	dev/cua0
COM 3	dev/ttyS0	dev/cua0
COM 4	dev/ttyS0	dev/cua0

Tabla 14: Tabla de entradas y salidas de los puertos serie en Linux

Al querer enviar la información vía serie USB, las direcciones que se deben buscar son:

- /dev/ttyUSB0
- /dev/ttyUSB1: se utiliza para audio
- /dev/ttyUSB2: se utiliza para enviar comandos AT

Por lo tanto, se debe seguir el ejemplo de la Figura 115, enviando al puerto ttyUSB2 los comandos AT específicos de resetear modem, escribir número de teléfono y escribir mensaje.

9. RESULTADOS FINALES

El interfaz de usuario que se muestra tras encender la aplicación se muestra en la Figura 116. En esta pantalla se muestran varios botones, con las siguientes funciones

- **START**: comienza el registro de datos con la configuracion que se ha establecido en el “Menu Configuracion”.
- **STOP**: se detiene el registro de datos
- **CONFIGURACION**: Se abre un menu para configurar la programacion temporal de cada canal de registro. Ademas se puede configurar en los cuales, si se supera, se enviara un mensaje de alarma al correo establecido por el usuario.
- **Mostrar Datos**: se expande la pantalla inicial y muestra los ultimos cuatro datos recibidos de cada canal.
- **Email**: permite escribir el email al cual se deberan mandar las copias de seguridad y cada cuanto tiempo.

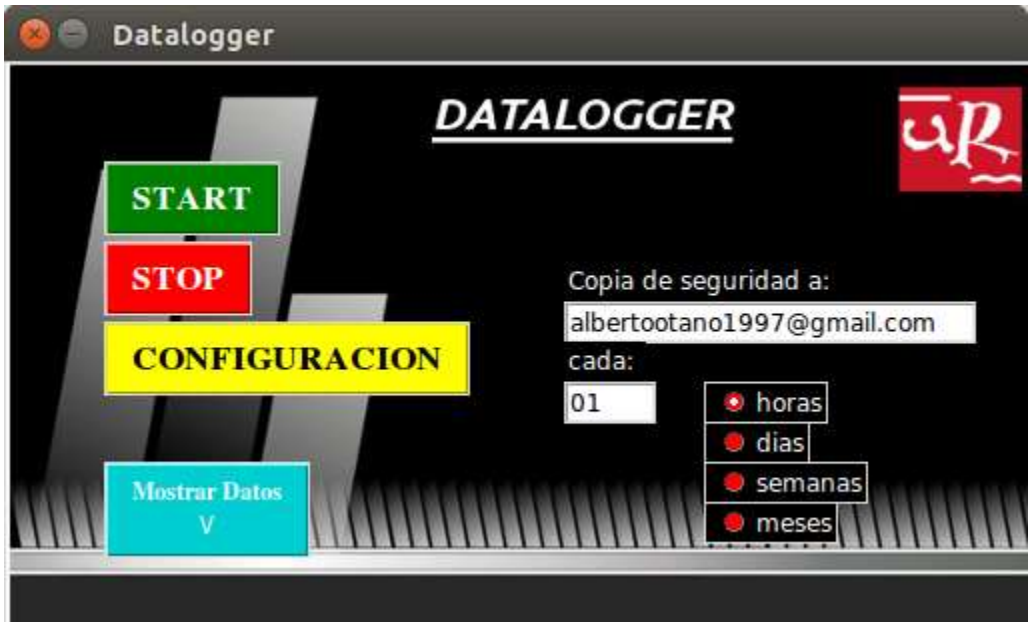


Figura 116:Interfaz de usuario inicial

Al pulsar en el botón CONFIGURACIÓN se abrirá la siguiente pantalla que permite programar temporalmente el muestreo (Figura 117). Además, permite establecer los umbrales de temperatura, humedad relativa y cantidad de lluvia entre los que deben estar los registros. En caso de sobrepasarse por encima o por debajo, se enviará un mensaje de alarma.

La aplicación tiene un sistema de salvaguardado que puede configurar el usuario, para evitar que por cualquier fallo del sistema se pierda el registro de datos. El usuario deberá establecer el correo electrónico al cual se van a enviar los datos y cada cuanto tiempo quiere que se envíen.

Figura 117: Menú Configuración

Se pueden ver los últimos cuatro datos adquiridos en cada canal. Para ello hay que pulsar el botón “Mostrar Datos” y se expandirá la pantalla de inicio tomando la apariencia de la 118.

Figura 118: Pantalla de la muestra de datos

Para comprobar que estos datos son correctos, se ha abierto el Terminal de Linux para ver los datos en formato hexadecimal, decimal convertido a milivoltios, decimal en las unidades correspondientes. El archivo capta 12 valores en total (4 de cada canal) y los agrupa por canales para devolver el valor promedio de cada canal, aunque tan solo será visible el valor del canal que corresponda en cada momento. Como se muestra en la Tabla 10: Agrupación de los valores por canales, los 4 LSB indican el canal al que corresponde el valor muestreado, siendo:

- **3:** el canal VPVN (correspondiente a Temperatura)
- **0:** el canal Vaux0 (correspondiente a Humedad Relativa)
- **8:** el canal Vaux8 (correspondiente a Lluvia).

En la siguiente Figura 119 se muestra un ejemplo de lo explicado en el citado párrafo.

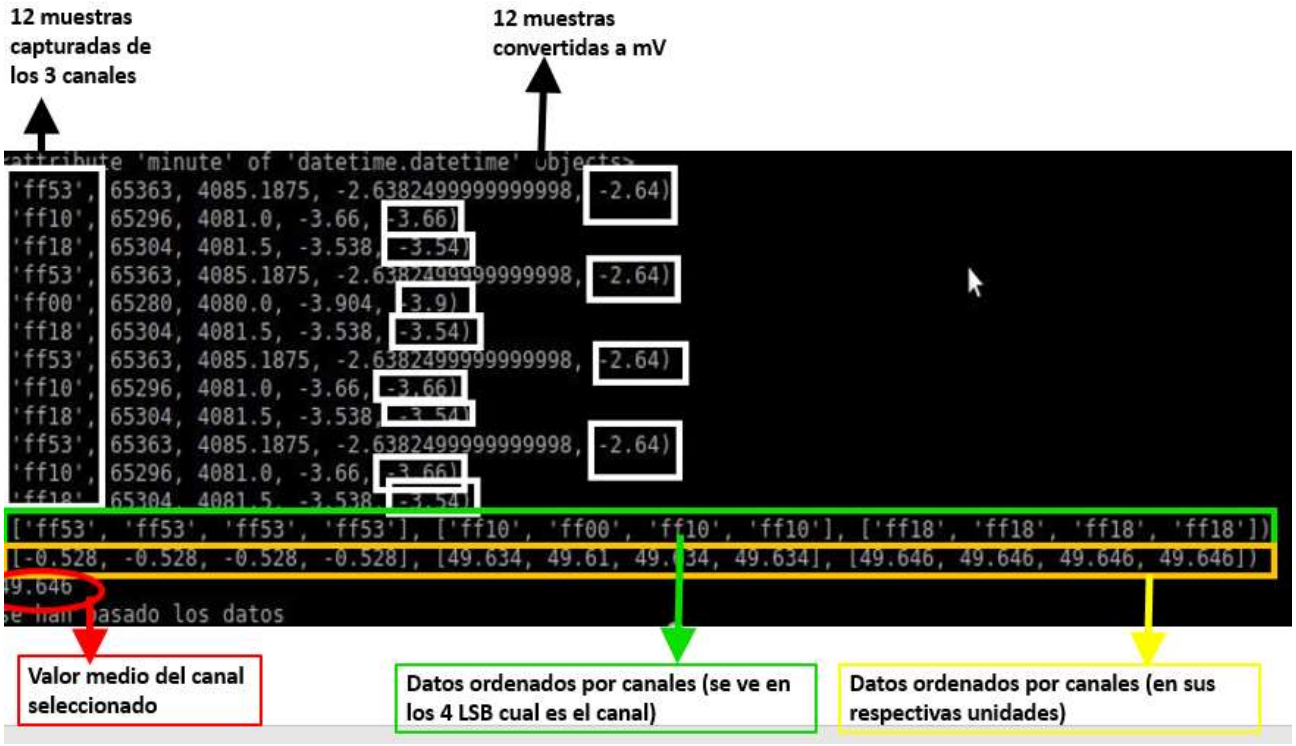


Figura 119: Muestra y agrupación de los datos captados en hexadecimal y decimal

Estos datos se guardan en formato .csv para que el usuario pueda operar con los datos, hacer gráficas, obtener promediados, etc.

Figura 120: Ficheros de salida del Datalogger

A continuación, se muestra un ejemplo de una captura de datos aleatorios, de cómo se exportan dichos datos a Excel y como se pueden realizar gráficas, hacer promediados o cualquier otra función estadística.

Ano	Mes	Dia	Hora	Minuto	Dato	Unidades
2019	6	26	18	14	-0,55	C
2019	6	26	18	15	-0,54	C
2019	6	26	18	16	-0,55	C
2019	6	26	18	17	-0,5	C
2019	6	26	18	18	-0,55	C
2019	6	26	18	19	-0,58	C
2019	6	26	18	21	-0,53	C
2019	6	26	18	22	-0,53	C
2019	6	26	18	23	-0,58	C
2019	6	26	18	24	-0,53	C
2019	6	26	18	25	-0,53	C
2019	6	26	18	26	-0,58	C
2019	6	26	18	27	-0,58	C
2019	6	26	18	29	-0,53	C

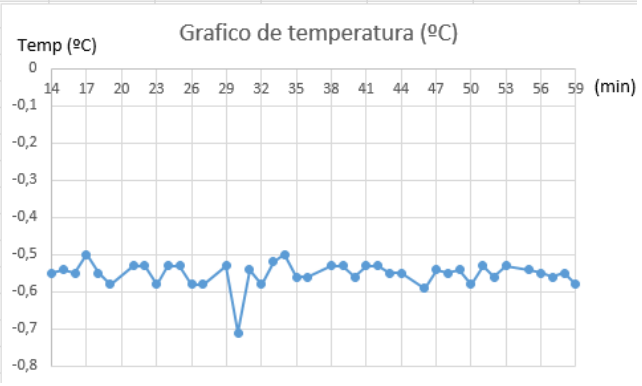


Figura 121: Ejemplo de captura de Temperatura en Excel

Ano	Mes	Dia	Hora	Minuto	Dato	Unidades
2019	6	26	18	14	49,634	%
2019	6	26	18	15	49,610	%
2019	6	26	18	16	49,610	%
2019	6	26	18	17	49,598	%
2019	6	26	18	18	49,585	%
2019	6	26	18	20	49,591	%
2019	6	26	18	21	49,591	%
2019	6	26	18	22	49,585	%
2019	6	26	18	23	49,585	%
2019	6	26	18	24	49,604	%
2019	6	26	18	25	49,610	%
2019	6	26	18	26	49,579	%
2019	6	26	18	27	49,598	%
2019	6	26	18	29	49,591	%
2019	6	26	18	30	49,604	%

Figura 122: Ejemplo de captura de Humedad Relativa en Excel

Ano	Mes	Dia	Hora	Minuto	Dato	Unidades
2019	6	26	18	14	49,646	mm
2019	6	26	18	15	49,634	mm
2019	6	26	18	16	49,652	mm
2019	6	26	18	17	49,712	mm
2019	6	26	18	18	49,646	mm
2019	6	26	18	20	49,622	mm
2019	6	26	18	21	49,628	mm
2019	6	26	18	22	49,622	mm
2019	6	26	18	23	49,730	mm
2019	6	26	18	24	49,736	mm
2019	6	26	18	25	49,622	mm
2019	6	26	18	26	49,622	mm
2019	6	26	18	28	49,622	mm
2019	6	26	18	29	49,628	mm
2019	6	26	18	30	49,628	mm
2019	6	26	18	31	49,634	mm

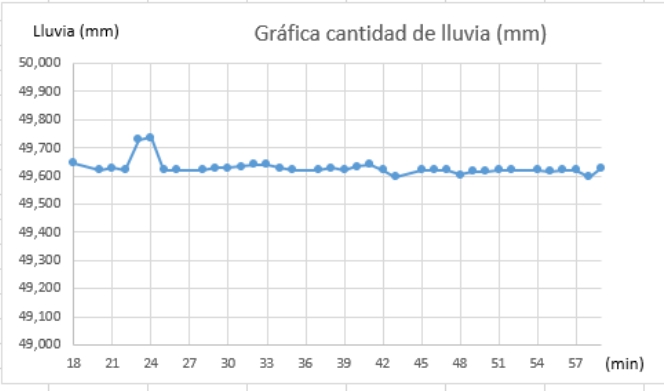


Figura 123: Ejemplo de captura de Cantidad de Lluvia en Excel

10. CONCLUSIONES

En el presente trabajo se han logrado los objetivos tanto generales como específicos establecidos al inicio. Se ha conseguido desarrollar una aplicación grafica de

usuario en alto nivel que sea capaz de comunicarse con el hardware del conversor analógico digital de la ZedBoard.

Una parte importante de este trabajo ha sido el estudio de la documentación técnica tanto de la ZedBoard como de la Zynq-7000 ya que, sin este estudio previo, hubiera sido imposible realizar cualquier desarrollo técnico. Por ejemplo, ha sido necesario conocer los registros donde almacena la información el XADC, los modos de funcionamiento, la parte PS y PL de la FPGA, información sobre Linux, etc. Aunque parezca que esta parte previa no haya tenido trascendencia en el proyecto, ha sido muy relevante para realizar el mismo.

En cuanto a la tarjeta elegida ZedBoard, se podría haber utilizado cualquier otra de menores prestaciones y por tanto menor coste, que sea capaz de leer un sistema operativo, como por ejemplo UltraZed. Al utilizar ZedBoard, se aseguran todas las funciones requeridas para este proyecto, pero se desaprovechan otras como la salida y entrada de audio, el uso de Switch y Leds, etc.

El desarrollo en Vivado ha sido complejo, ya que se partía de una cierta base aprendida durante la carrera, pero el proyecto requería un conocimiento ampliamente superior. Además, hoy en día las FPGAs tienen menor auge que los microprocesadores, por lo que el lenguaje de programación de estas (HDL) es menos utilizado y más difícil encontrar referencias. Sin embargo, gracias a los PSoC hay abierto un camino de aplicaciones para los sistemas embebidos, ya que unen la capacidad de desarrollo de la parte hardware con las posibilidades que ofrecen los sistemas operativos como Linux. Combinando ambas partes, las aplicaciones y proyectos que se pueden desarrollar con FPGAs son prácticamente ilimitadas.

La parte de comunicación entre la parte la parte de la FPGA y del Host (sistema operativo Linux) ha sido posible gracias a Xillybus, que permite la creación y descarga de IP Cores a la medida del desarrollador.

En lugar de trabajar con sistemas operativos, también se podría haber gestionado desde el hardware la parte de sincronismo de la pantalla, del USB, etc. Sin embargo, se ha utilizado esta opción para facilitarnos el control de estas funciones, ya que Linux ofrece estas funciones y, además, se puede gestionar la hora y la conexión a internet, que son imprescindibles para este proyecto.

Este proyecto también requería conocer el funcionamiento del sistema operativo Linux y de creación de aplicaciones con interfaz de usuario, así como el lenguaje de alto nivel Python. La curva de aprendizaje y desarrollo del proyecto ha sido lenta y costosa, ya que no se tenían conocimientos previos sobre ninguno de estos temas.

11. VÍAS DE CONTINUACIÓN

Como continuación del trabajo con el fin de mejorar el mismo quedarían entre otras mejoras, las siguientes que se proponen a continuación:

- Implementación del sistema con la colocación en una estación meteorológica para que se pueda utilizar como un *Datalogger* real y no como un prototipo.
- La colocación de los sensores apropiados y su correcta calibración utilizando sus correspondientes funciones de transferencia.
- Utilización del flujo de datos en la dirección Host-FPGA que se creó en el sistema, pero no se le ha dado ningún uso. Por ejemplo, mediante esta opción el usuario podría cambiar el registro de datos que se quiera en cada momento.
- Ampliación de los canales de adquisición de datos, en lugar de poder muestrear tres canales, que se pudieran muestrear hasta diecisiete. Para ello habría que utilizar el modo de multiplexor externo, ya que solo existen entradas analógicas físicas para los canales VpVn, Vaux0 y Vaux8.
- Almacenamiento de datos en bases de datos remotas, IoT, etc.
- Alarmas de desconexión de red, o motor de base de datos caído, almacenamiento en local y aviso de emergencia ante pérdida de datos.
- Auto chequeo de diagnóstico del sistema y generación de informes/alarmas cada vez que inicie y periódicamente, por ejemplo, cada 2horas,
- Exportación a diversos formatos de software de tratamiento de datos
- Seguridad y privacidad de los datos frente a posibles robos o intrusiones
- Control remoto de los parámetros de configuración mediante un dispositivo móvil o un ordenador personal.
- Envío de SMS, además de email, a modo de alarma cuando se hayan sobrepasado los umbrales establecidos por el usuario.

Envío de SMS a modo de alerta de que ha fallado la conexión Ethernet, o que el dispositivo de almacenamiento le queda poca capacidad de memoria. Estas funciones se explican de forma más detallada en el apartado 8.4. Otras funciones

UNIVERSIDAD DE LA RIOJA

Trabajo Fin de Grado

Desarrollo de un “Datalogger” basado en ARM Cortex A9 y Linux mediante dispositivo Zynq-7000

3. ANEXOS

Alberto Otaño Jiménez

Julio 2019

ANEXOS

1. Código VHDL

1.1. xillydemo.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
entity xillydemo is
port (                          --puerto de entradas/salidas de xillybus
                                --estas E/S se deben asociar a las constrains
clk_100 : IN std_logic;
otg_oc : IN std_logic;
PS_GPIO : INOUT std_logic_vector(55 DOWNTO 0);
GPIO_LED : OUT std_logic_vector(3 DOWNTO 0);
vga4_blue : OUT std_logic_vector(3 DOWNTO 0);
vga4_green : OUT std_logic_vector(3 DOWNTO 0);
vga4_red : OUT std_logic_vector(3 DOWNTO 0);
vga_hsync : OUT std_logic;
vga_vsync : OUT std_logic;
audio_mclk : OUT std_logic;
audio_dac : OUT std_logic;
audio_adc : IN std_logic;
audio_bclk : IN std_logic;
audio_lrclk : IN std_logic;
smb_sclk : OUT std_logic;
smb_sdata : INOUT std_logic;
smbus_addr : OUT std_logic_vector(1 DOWNTO 0);
-- entradas para el XADC
vp_in : IN STD_LOGIC;
vn_in : IN STD_LOGIC;
vauxp0 : IN STD_LOGIC;
vauxn0 : IN STD_LOGIC;
vauxp8 : IN STD_LOGIC;
vauxn8 : IN STD_LOGIC
);

end xillydemo;
architecture sample_arch of xillydemo is
component xillybus                    --DECLARACION DE COMPONENTES: Xillybus
port (
PS_CLK : IN std_logic;
PS_PORB : IN std_logic;
PS_SRSTB : IN std_logic;
clk_100 : IN std_logic;
otg_oc : IN std_logic;
DDR_Addr : INOUT std_logic_vector(14 DOWNTO 0);
DDR_BankAddr : INOUT std_logic_vector(2 DOWNTO 0);
DDR_CAS_n : INOUT std_logic;
DDR_CKE : INOUT std_logic;
DDR_CS_n : INOUT std_logic;
DDR_Clk : INOUT std_logic;
```

```
DDR_Clk_n : INOUT std_logic;
DDR_DM : INOUT std_logic_vector(3 DOWNTO 0);
DDR_DQ : INOUT std_logic_vector(31 DOWNTO 0);
DDR_DQS : INOUT std_logic_vector(3 DOWNTO 0);
DDR_DQS_n : INOUT std_logic_vector(3 DOWNTO 0);
DDR_DRSTB : INOUT std_logic;
DDR_ODT : INOUT std_logic;
DDR_RAS_n : INOUT std_logic;
DDR_VRN : INOUT std_logic;
DDR_VRP : INOUT std_logic;
MIO : INOUT std_logic_vector(53 DOWNTO 0);
PS_GPIO : INOUT std_logic_vector(55 DOWNTO 0);
DDR_WEB : OUT std_logic;
GPIO_LED : OUT std_logic_vector(3 DOWNTO 0);
bus_clk : OUT std_logic;
quiesce : OUT std_logic;
vga4_blue : OUT std_logic_vector(3 DOWNTO 0);
vga4_green : OUT std_logic_vector(3 DOWNTO 0);
vga4_red : OUT std_logic_vector(3 DOWNTO 0);
vga_hsync : OUT std_logic;
vga_vsync : OUT std_logic;
user_w_controlstream_wren : OUT std_logic;
user_w_controlstream_full : IN std_logic;
user_w_controlstream_data : OUT std_logic_vector(7 DOWNTO 0);
user_w_controlstream_open : OUT std_logic;
user_r_datastream_rden : OUT std_logic;
user_r_datastream_empty : IN std_logic;
user_r_datastream_data : IN std_logic_vector(15 DOWNTO 0);
user_r_datastream_eof : IN std_logic;
user_r_datastream_open : OUT std_logic;
user_clk : OUT std_logic;
user_wren : OUT std_logic;
user_rden : OUT std_logic;
user_wstrb : OUT std_logic_vector(3 DOWNTO 0);
user_addr : OUT std_logic_vector(31 DOWNTO 0);
user_rd_data : IN std_logic_vector(31 DOWNTO 0);
user_wr_data : OUT std_logic_vector(31 DOWNTO 0);
user_irq : IN std_logic);
end component;

component top_xadc is                    --DECLARACION DE COMPONENTES: XADC
port (
clk : IN std_logic;
reset : IN std_logic;
vp_in : IN STD_LOGIC;
vn_in : IN STD_LOGIC;
vauxp0 : IN STD_LOGIC;
vauxn0 : IN STD_LOGIC;
vauxp8 : IN STD_LOGIC;
vauxn8 : IN STD_LOGIC;
jtagLMB : OUT std_logic_vector(2 downto 0);
dout : OUT std_logic_vector(15 downto 0);
drdy : OUT std_logic;
channel : OUT std_logic_vector(4 downto 0)
);
```

```
end component;

component fifo is               --DECLARACION DE COMPONENTES: FIFO
port (
clk : IN STD_LOGIC;
srst : IN STD_LOGIC;
din : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
wr_en : IN STD_LOGIC;
rd_en : IN STD_LOGIC;
dout : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
full : OUT STD_LOGIC;
empty : OUT STD_LOGIC
);
end component;

signal bus_clk : std_logic;            --CREACION DE SEÑALES AUXILIARES
signal quiesce : std_logic;
signal user_w_controlstream_wren : std_logic;
signal user_w_controlstream_full : std_logic;
signal user_w_controlstream_data : std_logic_vector(7 DOWNTO 0);
signal user_w_controlstream_open : std_logic;
signal user_r_datastream_rden : std_logic;
signal user_r_datastream_empty : std_logic;
signal user_r_datastream_data : std_logic_vector(15 DOWNTO 0);
signal user_r_datastream_eof : std_logic;
signal user_r_datastream_open : std_logic;
signal user_clk : std_logic;
signal user_wren : std_logic;
signal user_rden : std_logic;
signal user_wstrb : std_logic_vector(3 DOWNTO 0);
signal user_addr : std_logic_vector(31 DOWNTO 0);
signal user_rd_data : std_logic_vector(31 DOWNTO 0);
signal user_wr_data : std_logic_vector(31 DOWNTO 0);
signal user_irq : std_logic;
-- Note that none of the ARM processor's direct connections to pads is
-- defined as I/O on this module. Normally, they should be connected
-- as toplevel ports here, but that confuses Vivado 2013.4 to think that
-- some of these ports are real I/Os, causing an implementation failure.
-- This detachment results in a lot of warnings during synthesis and
-- implementation, but has no practical significance, as these pads are
-- completely unrelated to the FPGA bitstream.
signal PS_CLK : std_logic;
signal PS_PORB : std_logic;
signal PS_SRSTB : std_logic;
signal DDR_Addr : std_logic_vector(14 DOWNTO 0);
signal DDR_BankAddr : std_logic_vector(2 DOWNTO 0);
signal DDR_CAS_n : std_logic;
signal DDR_CKE : std_logic;
signal DDR_CS_n : std_logic;
signal DDR_Clk : std_logic;
signal DDR_Clk_n : std_logic;
signal DDR_DM : std_logic_vector(3 DOWNTO 0);
signal DDR_DQ : std_logic_vector(31 DOWNTO 0);
signal DDR_DQS : std_logic_vector(3 DOWNTO 0);
signal DDR_DQS_n : std_logic_vector(3 DOWNTO 0);
```

```
signal DDR_DRSTB : std_logic;
signal DDR_ODT : std_logic;
signal DDR_RAS_n : std_logic;
signal DDR_VRN : std_logic;
signal DDR_VRP : std_logic;
signal MIO : std_logic_vector(53 DOWNTO 0);
signal DDR_WEB : std_logic;
-- señal para FIFO
signal reset_fifo : std_logic;
-- señales auxiliares entre XADC y FIFO
signal xadc_dout_fifo_din : std_logic_vector(15 downto 0);
signal xadc_drdy_fifo_wren : std_logic;

signal xadc_to_fifo_mas_canal:std_logic_vector (15 downto 0);
signal channel_aux: std_logic_vector(4 downto 0);

begin
xillybus_ins : xillybus         --Instanciacion de componentes: Xillybus
port map (
-- Ports related to /dev/xillybus_controlstream
-- CPU to FPGA signals:
user_w_controlstream_wren => user_w_controlstream_wren,
user_w_controlstream_full => user_w_controlstream_full,
user_w_controlstream_data => user_w_controlstream_data,
user_w_controlstream_open => user_w_controlstream_open,
-- Ports related to /dev/xillybus_datastream
-- FPGA to CPU signals:
user_r_datastream_rden => user_r_datastream_rden,
user_r_datastream_empty => user_r_datastream_empty,
user_r_datastream_data => user_r_datastream_data,
user_r_datastream_eof => user_r_datastream_eof,
user_r_datastream_open => user_r_datastream_open,
-- Ports related to Xillybus Lite
user_clk => user_clk,
user_wren => user_wren,
user_rden => user_rden,
user_wstrb => user_wstrb,
user_addr => user_addr,
user_rd_data => user_rd_data,
user_wr_data => user_wr_data,
user_irq => user_irq,
-- General signals
PS_CLK => PS_CLK,
PS_PORB => PS_PORB,
PS_SRSTB => PS_SRSTB,
clk_100 => clk_100,
otg_oc => otg_oc,
DDR_Addr => DDR_Addr,
DDR_BankAddr => DDR_BankAddr,
DDR_CAS_n => DDR_CAS_n,
DDR_CKE => DDR_CKE,
DDR_CS_n => DDR_CS_n,
DDR_Clk => DDR_Clk,
DDR_Clk_n => DDR_Clk_n,
DDR_DM => DDR_DM,
```

```
DDR_DQ => DDR_DQ,
DDR_DQS => DDR_DQS,
DDR_DQS_n => DDR_DQS_n,
DDR_DRSTB => DDR_DRSTB,
DDR_ODT => DDR_ODT,
DDR_RAS_n => DDR_RAS_n,
DDR_VRN => DDR_VRN,
DDR_VRP => DDR_VRP,
MIO => MIO,
PS_GPIO => PS_GPIO,
DDR_WEB => DDR_WEB,
GPIO_LED => GPIO_LED,
bus_clk => bus_clk,
quiesce => quiesce,
vga4_blue => vga4_blue,
vga4_green => vga4_green,
vga4_red => vga4_red,
vga_hsync => vga_hsync,
vga_vsync => vga_vsync
);

xadc : top_xadc           -- Instanciacion de componentes: XADC
port map (                --se conecta XADC con la FIFO
clk => bus_clk,
reset => '0',
vp_in => vp_in,
vn_in => vn_in,
vauxp0 => vauxp0,
vauxn0 => vauxn0,
vauxp8 => vauxp8,
vauxn8 => vauxn8,
jtagLMB => open,
dout => xadc_dout_fifo_din,     --salida del xadc con entrada a la fifo
drdy => xadc_drdy_fifo_wren,
channel => channel_aux
);

fifo16 : fifo
port map (
clk => bus_clk,
srst => reset_fifo,
din => xadc_dout_fifo_din,      --salida del xadc con entrada a la fifo
wr_en => xadc_drdy_fifo_wren,
rd_en => user_r_datastream_rden,
dout => user_r_datastream_data, --salida de la fifo con entrada a xillybus
full => open,
empty => user_r_datastream_empty
);
-- reseteamos la FIFO siempre que no se esté leyendo
reset_fifo <= not(user_r_datastream_open);
-- nunca se alcanza el end of file
user_r_datastream_eof <= '0';
end sample_arch;
```

1.2. top_xadc.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity top_xadc is
Port (clk : IN std_logic;       --Puerto de E/S del XADC
reset : IN std_logic;
vp_in : IN STD_LOGIC;
vn_in : IN STD_LOGIC;
vauxp0 : IN STD_LOGIC;
vauxn0 : IN STD_LOGIC;
vauxp8 : IN STD_LOGIC;
vauxn8 : IN STD_LOGIC;
jtagLMB : OUT std_logic_vector(2 downto 0);
dout : OUT std_logic_vector(15 downto 0);
drdy : OUT std_logic;
channel : OUT std_logic_vector(4 downto 0)
);
end top_xadc;

architecture Behavioral of top_xadc is

component xadc_wiz_0 is         --Declaracion del componente XADC
port ( daddr_in : in STD_LOGIC_VECTOR (6 downto 0);
den_in : in STD_LOGIC;
di_in : in STD_LOGIC_VECTOR (15 downto 0);
dwe_in : in STD_LOGIC;
do_out : out STD_LOGIC_VECTOR (15 downto 0);
drdy_out : out STD_LOGIC;
dclk_in : in STD_LOGIC;
reset_in : in STD_LOGIC;
jtagbusy_out : out STD_LOGIC;
jtaglocked_out : out STD_LOGIC;
jtagmodified_out : out STD_LOGIC;
vauxp0 : in STD_LOGIC;
vauxn0 : in STD_LOGIC;
vauxp8 : in STD_LOGIC;
vauxn8 : in STD_LOGIC;
busy_out : out STD_LOGIC;
channel_out : out STD_LOGIC_VECTOR (4 downto 0);
eoc_out : out STD_LOGIC;
eos_out : out STD_LOGIC;
alarm_out : out STD_LOGIC;
vp_in : in STD_LOGIC;
vn_in : in STD_LOGIC
);
```

```
end component;

-- señales auxiliares
signal aux_dout : std_logic_vector(15 downto 0);
signal aux_channel : std_logic_vector(4 downto 0);
signal aux_eoc : std_logic;
signal aux_daddr : std_logic_vector(6 downto 0);
signal senal_cortada: std_logic_vector (15 downto 0);

begin

xadc : xadc_wiz_0
port map (
daddr_in => aux_daddr,
den_in => aux_eoc,
di_in => (others => '0'),
dwe_in => '0',
do_out => aux_dout,
drdy_out => drdy,
dclk_in => clk,
reset_in => reset,
jtagbusy_out => jtagLMB(0),
jtaglocked_out => jtagLMB(2),
jtagmodified_out => jtagLMB(1),
vauxp0 => vauxp0,
vauxn0 => vauxn0,
vauxp8 => vauxp8,
vauxn8 => vauxn8,
busy_out => open,
channel_out => aux_channel,
eoc_out => aux_eoc,
eos_out => open,
alarm_out => open,
vp_in => vp_in,
vn_in => vn_in
);

-- puesto que daddr es de 7 bits y channel es de 5,
-- los 2 primeros bits serán siempre 0
aux_daddr <= "00" & aux_channel;
-- xillybus invierte el orden de los bytes, por eso
-- hay que poner primero el byte menos significativo
-- hasta el más significativo
senal_cortada <=aux_dout(15 downto 4) & aux_channel(3 downto 0);
dout <= senal_cortada(7 downto 0) & senal_cortada(15 downto 8);
-- se conecta la señal
channel <= aux_channel;

end Behavioral;
```

1.3. Archivo de restricciones (constrains): xillydemo.xdc

```
create_clock -name gclk -period 10 [get_ports "clk_100"]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets "clk_100"]
```

```
# Vivado constraints unrelated clocks. So set false paths.
set_false_path -from [get_clocks clk_fpga_1] -to [get_clocks vga_clk_ins/*]
set_false_path -from [get_clocks vga_clk_ins/*] -to [get_clocks clk_fpga_1]
# The VGA outputs are turned into an analog voltage by virtue of a resistor
# network, so the flip flops driving these must sit in the IOBs to minimize
# timing skew. The RTL code should handle this, but the constraint below
# is there to fail if something goes wrong about this.
set_output_delay 5.5 [get_ports {vga*}]
set_property -dict "PACKAGE_PIN Y9 IOSTANDARD LVCMOS33" [get_ports "clk_100"]
set_property -dict "PACKAGE_PIN T22 IOSTANDARD LVCMOS33" [get_ports
"GPIO_LED[0]"]
set_property -dict "PACKAGE_PIN T21 IOSTANDARD LVCMOS33" [get_ports
"GPIO_LED[1]"]
set_property -dict "PACKAGE_PIN U22 IOSTANDARD LVCMOS33" [get_ports
"GPIO_LED[2]"]

set_property -dict "PACKAGE_PIN U21 IOSTANDARD LVCMOS33" [get_ports
"GPIO_LED[3]"]
set_property -dict "PACKAGE_PIN Y21 IOSTANDARD LVCMOS33" [get_ports
"vga4_blue[0]"]
set_property -dict "PACKAGE_PIN Y20 IOSTANDARD LVCMOS33" [get_ports
"vga4_blue[1]"]
set_property -dict "PACKAGE_PIN AB20 IOSTANDARD LVCMOS33" [get_ports
"vga4_blue[2]"]
set_property -dict "PACKAGE_PIN AB19 IOSTANDARD LVCMOS33" [get_ports
"vga4_blue[3]"]
set_property -dict "PACKAGE_PIN AB22 IOSTANDARD LVCMOS33" [get_ports
"vga4_green[0]"]
set_property -dict "PACKAGE_PIN AA22 IOSTANDARD LVCMOS33" [get_ports
"vga4_green[1]"]
set_property -dict "PACKAGE_PIN AB21 IOSTANDARD LVCMOS33" [get_ports
"vga4_green[2]"]
set_property -dict "PACKAGE_PIN AA21 IOSTANDARD LVCMOS33" [get_ports
"vga4_green[3]"]
set_property -dict "PACKAGE_PIN V20 IOSTANDARD LVCMOS33" [get_ports
"vga4_red[0]"]
set_property -dict "PACKAGE_PIN U20 IOSTANDARD LVCMOS33" [get_ports
"vga4_red[1]"]
set_property -dict "PACKAGE_PIN V19 IOSTANDARD LVCMOS33" [get_ports
"vga4_red[2]"]
set_property -dict "PACKAGE_PIN V18 IOSTANDARD LVCMOS33" [get_ports
"vga4_red[3]"]
set_property -dict "PACKAGE_PIN Y19 IOSTANDARD LVCMOS33" [get_ports
"vga_vsync"]
set_property -dict "PACKAGE_PIN AA19 IOSTANDARD LVCMOS33" [get_ports
"vga_hsync"]
# IMPORTANT: Since four LEDs are taken by the Xillybus IP core, the pin
# placement doesn't match the one given by Digilent.
# GPIO pin to reset the USB OTG PHY
set_property -dict "PACKAGE_PIN G17 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[0]"]
# On-board OLED
set_property -dict "PACKAGE_PIN U11 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[1]"]
```

```
set_property -dict "PACKAGE_PIN U12 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[2]"]
set_property -dict "PACKAGE_PIN U9 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[3]"]
set_property -dict "PACKAGE_PIN U10 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[4]"]
set_property -dict "PACKAGE_PIN AB12 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[5]"]
set_property -dict "PACKAGE_PIN AA12 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[6]"]
# On-board LEDs. Note that only for LEDs are allocated, as opposed to
# Digilent's eight, and all placements that follow are shifted by four.
# There was no other choice, as the tools don't allow unplaced PS GPIO pins.
set_property -dict "PACKAGE_PIN V22 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[7]"]
set_property -dict "PACKAGE_PIN W22 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[8]"]
set_property -dict "PACKAGE_PIN U19 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[9]"]
set_property -dict "PACKAGE_PIN U14 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[10]"]
# On-board Slide Switches
set_property -dict "PACKAGE_PIN F22 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[11]"]
set_property -dict "PACKAGE_PIN G22 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[12]"]
set_property -dict "PACKAGE_PIN H22 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[13]"]
set_property -dict "PACKAGE_PIN F21 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[14]"]
set_property -dict "PACKAGE_PIN H19 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[15]"]
set_property -dict "PACKAGE_PIN H18 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[16]"]
set_property -dict "PACKAGE_PIN H17 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[17]"]
set_property -dict "PACKAGE_PIN M15 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[18]"]
# On-board Left, Right, Up, Down, and Select Pushbuttons
set_property -dict "PACKAGE_PIN N15 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[19]"]
set_property -dict "PACKAGE_PIN R18 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[20]"]
set_property -dict "PACKAGE_PIN T18 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[21]"]
set_property -dict "PACKAGE_PIN R16 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[22]"]
set_property -dict "PACKAGE_PIN P16 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[23]"]
# Pmod JA
set_property -dict "PACKAGE_PIN Y11 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[24]"]
set_property -dict "PACKAGE_PIN AA11 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[25]"]
```

```
set_property -dict "PACKAGE_PIN Y10 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[26]"]
set_property -dict "PACKAGE_PIN AA9 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[27]"]
set_property -dict "PACKAGE_PIN AB11 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[28]"]
set_property -dict "PACKAGE_PIN AB10 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[29]"]
set_property -dict "PACKAGE_PIN AB9 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[30]"]
set_property -dict "PACKAGE_PIN AA8 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[31]"]
# Pmod JB
set_property -dict "PACKAGE_PIN W12 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[32]"]
set_property -dict "PACKAGE_PIN W11 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[33]"]
set_property -dict "PACKAGE_PIN V10 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[34]"]
set_property -dict "PACKAGE_PIN W8 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[35]"]
set_property -dict "PACKAGE_PIN V12 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[36]"]
set_property -dict "PACKAGE_PIN W10 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[37]"]
set_property -dict "PACKAGE_PIN V9 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[38]"]
set_property -dict "PACKAGE_PIN V8 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[39]"]
# Pmod JC
set_property -dict "PACKAGE_PIN AB7 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[40]"]
set_property -dict "PACKAGE_PIN AB6 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[41]"]
set_property -dict "PACKAGE_PIN Y4 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[42]"]
set_property -dict "PACKAGE_PIN AA4 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[43]"]
set_property -dict "PACKAGE_PIN R6 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[44]"]
set_property -dict "PACKAGE_PIN T6 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[45]"]
set_property -dict "PACKAGE_PIN T4 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[46]"]
set_property -dict "PACKAGE_PIN U4 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[47]"]
# Pmod JD
set_property -dict "PACKAGE_PIN V7 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[48]"]
set_property -dict "PACKAGE_PIN W7 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[49]"]
set_property -dict "PACKAGE_PIN V5 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[50]"]
set_property -dict "PACKAGE_PIN V4 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[51]"]
```

```
set_property -dict "PACKAGE_PIN W6 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[52]"]
set_property -dict "PACKAGE_PIN W5 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[53]"]
set_property -dict "PACKAGE_PIN U6 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[54]"]
set_property -dict "PACKAGE_PIN U5 IOSTANDARD LVCMOS33" [get_ports
"PS_GPIO[55]"]
# Pin for detecting USB OTG over-current condition
set_property -dict "PACKAGE_PIN L16 IOSTANDARD LVCMOS33" [get_ports "otg_oc"]
# Pins connected to sound chip
set_property -dict "PACKAGE_PIN AB1 IOSTANDARD LVCMOS33" [get_ports
"smbus_addr[0]"]
set_property -dict "PACKAGE_PIN Y5 IOSTANDARD LVCMOS33" [get_ports
"smbus_addr[1]"]
set_property -dict "PACKAGE_PIN AB4 IOSTANDARD LVCMOS33" [get_ports
"smb_sclk"]
set_property -dict "PACKAGE_PIN AB5 IOSTANDARD LVCMOS33" [get_ports
"smb_sdata"]
set_property -dict "PACKAGE_PIN Y8 IOSTANDARD LVCMOS33" [get_ports
"audio_dac"]
set_property -dict "PACKAGE_PIN AA7 IOSTANDARD LVCMOS33" [get_ports
"audio_adc"]
set_property -dict "PACKAGE_PIN AA6 IOSTANDARD LVCMOS33" [get_ports
"audio_bclk"]
set_property -dict "PACKAGE_PIN Y6 IOSTANDARD LVCMOS33" [get_ports
"audio_lrclk"]
set_property -dict "PACKAGE_PIN AB2 IOSTANDARD LVCMOS33" [get_ports
"audio_mclk"]
# Pins para el XADC
set_property -dict "PACKAGE_PIN L11 IOSTANDARD LVCMOS33" [get_ports "vp_in"]
set_property -dict "PACKAGE_PIN M12 IOSTANDARD LVCMOS33" [get_ports "vn_in"]
set_property -dict "PACKAGE_PIN F16 IOSTANDARD LVCMOS33" [get_ports "vauxp0"]
set_property -dict "PACKAGE_PIN E16 IOSTANDARD LVCMOS33" [get_ports "vauxn0"]
set_property -dict "PACKAGE_PIN D16 IOSTANDARD LVCMOS33" [get_ports "vauxp8"]
set_property -dict "PACKAGE_PIN D17 IOSTANDARD LVCMOS33" [get_ports "vauxn8"]
set_property BITSTREAM.GENERAL.JTAG_XADC Disable [current_design]
```

2. Código Python

```
#!/usr/bin/python   #para que Linux lo interprete como script
ejecutable
# -*- coding: utf-8 -*-

#IMPORTAMOS LIBRERIAS
import matplotlib as mpl
import numpy as np
import sys
from Tkinter import *
import Tkinter as tk
import time
import binascii
import os
import sys
sys.setrecursionlimit(1000000)
```

```
from datetime import date, datetime
import matplotlib.backends.tkagg as tkagg
from matplotlib.backends.backend_agg import FigureCanvasAgg
import csv
import statistics as stats
import threading
import Queue
import tkMessageBox
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.base import MIMEBase
from email import encoders

#CREACION DE VENTANA PRINCIPAL
ventana = tk.Tk()
ventana.title("Datalogger")
ventana.geometry("512x280")
ventana.config(bg="dark turquoise")
ventana.resizable(width=False, height=False)        #evitar que se
pueda agrandar--> quitar boton de maximizar
#CREACION DE VENTANA DE CONFIGURACION
vConfi =tk.Toplevel()
vConfi.geometry("1024x768")
vConfi.title("Menu Configuracion")
vConfi.withdraw()
vConfig=tk.Canvas(vConfi,width=1500, height=1000)
vConfig.place(x=0,y=0)
#INSERTAR FOTOS EN LAS PANTALLAS
imagenVCONFIG=PhotoImage(file="/root/Desktop/DATALOGGER2_tarjeta/fondo
.png")
fondo=Label(vConfig,image=imagenVCONFIG).place(x=0,y=0)
imagenPRINCIPAL=PhotoImage(file="/root/Desktop/DATALOGGER2_tarjeta/fon
doprincipal.png")
fondo=Label(ventana,image=imagenPRINCIPAL).place(x=0,y=0)

#Creacion de canvas donde van los modos de configuracion
canvas0 = tk.Canvas(vConfig, width=270, height=170)
canvas0.config(bg="white")
canvas0.place(x=290,y=120)
canvas1 = tk.Canvas(vConfig, width=270, height=170)
canvas1.config(bg="white")
canvas1.place(x=590,y=120)
canvas2 = tk.Canvas(vConfig, width=270, height=170)
canvas2.config(bg="white")
canvas2.place(x=290,y=320)
canvas3 = tk.Canvas(vConfig, width=270, height=170)
canvas3.config(bg="white")
canvas3.place(x=590,y=320)
canvas4 = tk.Canvas(vConfig, width=270, height=170)
canvas4.config(bg="white")
canvas4.place(x=290,y=520)
canvas5 = tk.Canvas(vConfig, width=270, height=170)
canvas5.config(bg="white")
canvas5.place(x=590,y=520)
#Creacion de canvas para mostrar datos
canvas10 = tk.Canvas(ventana, width=500, height=100)
canvas10.config(bg="white")
canvas10.place(x=10,y=380)
canvas20 = tk.Canvas(ventana, width=500, height=100)
canvas20.config(bg="white")
```

```
canvas20.place(x=10,y=540)
canvas30 = tk.Canvas(ventana, width=480, height=100)
canvas30.config(bg="white")
canvas30.place(x=520,y=380)
Label(canvas10,text="Ano          Mes          Dia          Hora
Minuto          Dato
Unidades",bg="gray").place(x=0,y=0)
Label(canvas20,text="Ano          Mes          Dia          Hora
Minuto          Dato
Unidades",bg="gray").place(x=0,y=0)
Label(canvas30,text="Ano          Mes          Dia          Hora
Minuto          Dato          Unidades",bg="gray").place(x=0,y=0)
Label(ventana,text='Temperatura',bg="yellow
green",fg="red",font=("Times", 12, "bold"),padx=205).place(x=10,y=355)
Label(ventana,text='Humedad relativa',bg="yellow
green",fg="red",font=("Times", 12, "bold"),padx=190).place(x=10,y=515)
Label(ventana,text='Lluvia',bg="yellow green",fg="red",font=("Times",
12, "bold"),padx=225).place(x=520,y=355)

#VARIABLES
#Variables int
minutoC0 = minutoC1 = minutoC2 = 0
horaC0=horaC1=horaC2=0
FIN=0
tPeriodoC0=tPeriodoC1=tPeriodoC2=0
tPerC0=tPerC1=tPerC2=0
PARAR=0
a=b=0
i0=i1=i2=0
MOSTRAR=0
ENVIAR=0
copia=1
tCopia=0
#Variables de los radiobutton, checkbutton, entrys, etc
selprogPeriodo0=IntVar()
selprogPeriodo1=IntVar()                    #si progPeriodo=1--> se
programa cada 30 min
selprogPeriodo2=IntVar()                   #si progPeriodo=0--> se
programa p.ej. LUNES a las 8
selMODIFICAR=IntVar()
selPARAR=IntVar()
selMOSTRAR=IntVar()
selENVIAR=IntVar()
selEmail=StringVar()
selCopia=IntVar()
#Variable string
email='albertootano1997@gmail.com'
#Creacion de cola para las interrupciones
cola = Queue.Queue()
#FUNCIONES
#Abrir pantalla de configuracion
def abrirConfig():
    vConfi.deiconify()
#Cerrar pantalla de configuracion
def Salir():
    vConfi.withdraw()
#Convertir valor hexadecimal a mV (segun fdt de XADC)
def conversion(value):
    valorint = int(value,16)
    valorfloat=float(valorint)
    valor = valorfloat/16
```

```
    if (valor <=2047):
        resultado=0.244*valor
    else:
        resultado=(4095-valor+1)*(-0.244)
    resultado2=round(resultado,2)
    print(value,valorint,valor,resultado,resultado2)
    return resultado
#COnvertir mV a Temp,Hum,Lluvia
def conversionVariable(valor,valormax,valormin):
    valorfloat=float(valor)
    floatmax=float(valormax)
    floatmin=float(valormin)
    pte=(floatmax - floatmin)/1000.00
    offset=floatmax - pte*500.00

    resultado = valorfloat*pte + offset
    return round (resultado,3)
#leer /dev/xillybus_datastream y crear un documento auxiliar con los
datos
def documentoAuxiliar():
    with open('/dev/xillybus_datastream', mode='rb') as file:
        fileContent = file.read(384)    #leer 384 bytes
        print(fileContent)
   file.close()

        with
open('/media/root/ALBERTO/DATALOGGER2_tarjeta/archivoXADC2.xadc',
mode='w') as file_wr:
            file_wr.write(fileContent)
#Obtener los datos ordenados del documento auxiliar
def CaptarDatos(canal):
    FIN=0
    documentoAuxiliar()
    while ( FIN==0):
        print (str(datetime.minute))
        with
open("/media/root/ALBERTO/DATALOGGER2_tarjeta/archivoXADC2.xadc",
mode='rb') as file:
            size_file = 24            #24 muestras
            #creacion de listas para cada canal
            vpvn = []
            vaux0 = []
            vaux8 = []
            vpvn_conv=[]
            vaux0_conv=[]
            vaux8_conv=[]
            count_channel = 0
            count = 0
            while (count < size_file):
                file.seek(count)    #seek es un puntero que apunta a
donde le indica count
                #Guardar datos del canal VPVN
                if count_channel == 0:
                    file.seek(count)
                    couple_bytes = file.read(2) #lee bytes de 2 en 2
                    hex_data = binascii.hexlify(couple_bytes)
                    vpvn.append(hex_data)
                    datomV=conversion(hex_data)
                    datoTemp=conversionVariable(datomV,100,-100)
                    vpvn_conv.append(datoTemp)  #append añade el dato
a la lista
```

```
                    count=count+2
                    count_channel = 1
                #Guardar datos del canal Vaux0
                elif count_channel == 1:
                    file.seek(count)
                    couple_bytes = file.read(2)
                    hex_data = binascii.hexlify(couple_bytes)
                    vaux0.append(hex_data)
                    datomV=conversion(hex_data)
                    datoHum=conversionVariable(datomV,100,0)
                    vaux0_conv.append(datoHum)
                    count=count+2
                    count_channel = 2
                #Guardar datos del canal Vaux8
                else:
                    file.seek(count)
                    couple_bytes = file.read(2)
                    hex_data = binascii.hexlify(couple_bytes)
                    vaux8.append(hex_data)
                    datomV=conversion(hex_data)
                    datoLluvia=conversionVariable(datomV,100,0)
                    vaux8_conv.append(datoLluvia)
                    count_channel = 0
                    count=count+2
            FIN=1
            #hace la media de la lista de cada canal
            mediaVPVN=stats.mean(vpvn_conv)
            mediaVAUX0=stats.mean(vaux0_conv)
            mediaVAUX8=stats.mean(vaux8_conv)
        print(vpvn,vaux0,vaux8)
        print(vpvn_conv,vaux0_conv,vaux8_conv)
              #segun el canal devuelve el valor promedio
            if canal==0:
                return mediaVPVN
            elif canal==1:
                return mediaVAUX0
            else:
                return mediaVAUX8

def STOP():
    global PARAR
    PARAR=1
    cola.put((selPARAR.set(1), ("Exito", "La persiana se ha cerrado"),
{}))
    #.set(valor)--> para añadir un valor a la IntVar()
    #.get()--> para obtener ese valor de la IntVar()
#Funciones para cambiar de tamaño la ventana principal
def MOSTRAR():
    global MOSTRAR
    ventana.geometry("1024x768")
    bMOSTRAR.place_forget()
    bOCULTAR.place(x=50,y=200)
    MOSTRAR=1
    cola.put((selMOSTRAR.set(1), ("Exito", "La persiana se ha
cerrado"), {}))
def OCULTAR():
    global OCULTAR
    ventana.geometry("512x280")
    bOCULTAR.place_forget()
    bMOSTRAR.place(x=50,y=200)
    MOSTRAR=0
```

```
        cola.put((selMOSTRAR.set(0), ("Exito", "La persiana se ha
    cerrado"), {}))

    #Funcion para enviar un email
    def enviarEmail():
        global email
        email=selEmail.get()
        x=datetime.now()
        print(x)
        fromaddr = 'TFGAlbertoOtano@gmail.com'
        toaddrs  = email
        asunto = ('Datos hasta la fecha: '+ str (x))
        cuerpo = ('Fecha de envio del mensaje:' + str (x))
        mensaje = MIMEMultipart()
        # Establecemos los atributos del mensaje
        mensaje['From'] = fromaddr
        mensaje['To'] = ", ".join(toaddrs)
        mensaje['Subject'] = asunto
        # Creamos el objeto mensaje
        ruta_adjunto0 =
    '/media/root/ALBERTO/DATALOGGER2_tarjeta/CSV/canal0.csv'
        nombre_adjunto0 = 'canalTemperatura.csv'
        ruta_adjunto1 =
    '/media/root/ALBERTO/DATALOGGER2_tarjeta/CSV/canal1.csv'
        nombre_adjunto1 = 'canalHumedad.csv'
        ruta_adjunto2 =
    '/media/root/ALBERTO/DATALOGGER2_tarjeta/CSV/canal2.csv'
        nombre_adjunto2 = 'canalLluvia.csv'
        # Agregamos el cuerpo del mensaje como objeto MIME de tipo texto
        mensaje.attach(MIMEText(cuerpo, 'plain'))
        # Abrimos el archivo que vamos a adjuntar
        archivo_adjunto0 = open(ruta_adjunto0, 'rb')
        archivo_adjunto1 = open(ruta_adjunto1, 'rb')
        archivo_adjunto2 = open(ruta_adjunto2, 'rb')
        # Creamos un objeto MIME base
        adjunto0_MIME = MIMEBase('application', 'octet-stream')
        adjunto1_MIME = MIMEBase('application', 'octet-stream')
        adjunto2_MIME = MIMEBase('application', 'octet-stream')
        # Y le cargamos el archivo adjunto
        adjunto0_MIME.set_payload((archivo_adjunto0).read())
        adjunto1_MIME.set_payload((archivo_adjunto1).read())
        adjunto2_MIME.set_payload((archivo_adjunto2).read())
        # Codificamos el objeto en BASE64
        encoders.encode_base64(adjunto0_MIME)
        encoders.encode_base64(adjunto1_MIME)
        encoders.encode_base64(adjunto2_MIME)
        # Agregamos una cabecera al objeto
        adjunto0_MIME.add_header('Content-Disposition', "attachment;
    filename= %s" % nombre_adjunto0)
        adjunto1_MIME.add_header('Content-Disposition', "attachment;
    filename= %s" % nombre_adjunto1)
        adjunto2_MIME.add_header('Content-Disposition', "attachment;
    filename= %s" % nombre_adjunto2)
        # Y finalmente lo agregamos al mensaje
        mensaje.attach(adjunto0_MIME)
        mensaje.attach(adjunto1_MIME)
        mensaje.attach(adjunto2_MIME)
        mensaje2 = mensaje.as_string()
        username = 'TFGAlbertoOtano@gmail.com'
        password = 'tfgalberto'
        #creacion del servidor
```

```
        server = smtplib.SMTP('smtp.gmail.com:587')
        server.starttls()
        server.login(username,password)
        server.sendmail(fromaddr, toaddrs,mensaje2)
        server.quit()

    def enviarAlarma(mensajeenviado):
        global email
        email=selEmail.get()
        x=datetime.now()
        fromaddr = 'TFGAlbertoOtano@gmail.com'
        toaddrs  = email
        asunto = ('Aviso de alarma con fecha: '+ str (x))
        cuerpo = mensajeenviado
        mensaje = MIMEMultipart()
        mensaje.attach(MIMEText(cuerpo, 'plain'))
        mensaje['From'] = fromaddr
        mensaje['To'] = ", ".join(toaddrs)
        mensaje['Subject'] = asunto
        mensaje2=mensaje.as_string()
        username = 'TFGAlbertoOtano@gmail.com'
        password = 'tfgalberto'
        server = smtplib.SMTP('smtp.gmail.com:587')
        server.starttls()
        server.login(username,password)
        server.sendmail(fromaddr, toaddrs, mensaje2)
        server.quit()

    #BUCLE WHILE PRINCIPAL (compara fecha y hora, capta datos y los
    escribe)
    def ContarTiempo():
        i0=i1=i2=0
        def run():      #funcion para interrupcion

    Label(ventana,text="corriendo...",bg="lime",font=("Helvetica",10,"ital
    ic")).place(x=250,y=65)
            ultimomin=0
            global
    tPerC0,tPerC1,tPerC2,a,b,PARAR,MOSTRAR,ENVIAR,email,tCopia,maxC0,minC0
    ,maxC1,minC1,maxC2,minC2
            convertirMaxMin()      #obtener umbrales max y min
            email=selEmail.get()     #obtener email
            copia=selCopia.get()     #obtener el tiempo de copia de
    seguridad
            tCopia=convertirTiempoCopia(copia)
            progPeriodo0=selprogPeriodo0.get()      #obtener modo de
    programacion
            progPeriodo1=selprogPeriodo1.get()
            progPeriodo2=selprogPeriodo2.get()
            t0=tPerC0
            t1=tPerC1
            t2=tPerC2
            tc=tCopia
            dato=[0,0,0]         #creacion de vector para guardar datos de
    los 3 canales
            selPARAR.set(0)      #poner PARAR a 0
            PARAR=selPARAR.get()
            canal=0
            x=datetime.now()     #obtener fecha
            anioactual=x.year    #descomponer fecha
            mesactual=x.month
```

```
        diaactual=x.day
        diasemanaactual=time.strftime("%A")
        horaactual=x.hour
        minactual= x.minute
        diasemana=convertirDias()   #obtener los dias seleccionados
por usuario
        matrizdias=[diasemana[0],diasemana[1],diasemana[2]]
        minuto=(minutoC0,minutoC1,minutoC2)
        hora=(horaC0,horaC1,horaC2)
        while(
            ((minactual!=minuto[a]) or (horaactual!=hora[a]) or
(diasemanaactual!=matrizdias[a][b]))or
            ((t0>0) or (t1>0) or (t2>0))
            and PARAR==0
            ):
            #mientras no sea la fecha establecida y no haya
transcurrido el tiempo de programacion
                for a in range (3):
                    #resta un segundo a cada canal
                    t0=t0-1
                    t1=t1-1
                    t2=t2-1
                    tc=tc-1
                    time.sleep(1)
                    if tc==0:
                        enviarEmail()
                        tc=tCopia
                    PARAR=selPARAR.get()    #comprobar a ver si se ha
pulsado STOP
                    print("parar",PARAR)
                    if PARAR==1:
                        print("ha salido")
                        break
                    MOSTRAR=selMOSTRAR.get()
                    if MOSTRAR==0:      #esta oculto la visualizacion
de datos
                       ventana.geometry("512x280")
                       bOCULTAR.place_forget()
                       bMOSTRAR.place(x=50,y=200)
                    else:               #esta visible la visualizacion
de datos
                        ventana.geometry("1024x768")
                        bMOSTRAR.place_forget()
                        bOCULTAR.place(x=50,y=200)
                    for b in range (7):
                        print (matrizdias[a][b])       #a= canal,
b=dia (l,m,x,j,v,s,d)
                        x=datetime.now()
                        diaactual=x.day
                        horaactual=x.hour
                        minactual= x.minute
                        print (x)
                        print (a,minuto[a])
                        print (t0,t1,t2)
                        print (tPerC0,tPerC1,tPerC2)
                        #si esta activo el modo progPeriodo y el
tiempo ha bajado a 0--> captar dato
                        if progPeriodo0==1:
                            if t0==0:
                                dato[0]=CaptarDatos(0)
                                GuardarDato(0,dato[0])
```

```
                                t0=tPerC0
                            if progPeriodo1==1:
                                if t1==0:
                                    dato[1]=CaptarDatos(1)
                                    GuardarDato(1,dato[1])
                                    t1=tPerC1
                            if progPeriodo2==1:
                                if t2==0:
                                    dato[2]=CaptarDatos(2)
                                    GuardarDato(2,dato[2])
                                    t2=tPerC2
                            #si la fecha y hora es la establecida-->
captar dato
                            if (((minactual==minuto[a]) and
(horaactual==hora[a]) and (diasemanaactual==matrizdias[a][b]))and
PARAR==0):
                                dato[a]=CaptarDatos(a)
                                if ultimomin!=minactual:            #para
que solo lo guarde una vez
                                    ultimomin=minactual
                                    GuardarDato(a,dato[a])
                    if PARAR==1:
                        print("ha salido2")
                Label(ventana,text="parado
",bg="red",font=("Helvetica",10,"italic")).place(x=250,y=65)
                time.sleep(2)
                Label(ventana,text="parado
",bg="black",font=("Helvetica",10,"italic")).place(x=250,y=65)
                    break
        #interrupcion
        t=threading.Thread(target=run)
        t.start()

    def tkloop():
        try:
            while True:
                funcion, args, kwargs  = cola.get_nowait()
                funcion(*args, **kwargs)
                time.sleep(1)
        except:
            pass
        ventana.after(100, tkloop)

    tkloop()
    #Etiquetas para poner la fecha y los datos
    label3C0=Label(canvas10,text=(""),bg="white")
    label2C0=Label(canvas10,text=(""),bg="white")
    label1C0=Label(canvas10,text=(""),bg="white")
    label0C0=Label(canvas10,text=(""),bg="white")
    label3C1=Label(canvas20,text=(""),bg="white")
    label2C1=Label(canvas20,text=(""),bg="white")
    label1C1=Label(canvas20,text=(""),bg="white")
    label0C1=Label(canvas20,text=(""),bg="white")
    label3C2=Label(canvas30,text=(""),bg="white")
    label2C2=Label(canvas30,text=(""),bg="white")
    label1C2=Label(canvas30,text=(""),bg="white")
    label0C2=Label(canvas30,text=(""),bg="white")
    #Funcion para guardar los datos
    def GuardarDato(canal,dato):
        global i0,i1,i2
        global
```

```
label0C0,label1C0,label2C0,label3C0,label0C1,label1C1,label2C1,label3C
1,label0C2,label1C2,label2C2,label3C2
    global maxC0,minC0,maxC1,minC1,maxC2,minC2
    if canal==0:
        archivo = open
("/media/root/ALBERTO/DATALOGGER2_tarjeta/CSV/canal0.csv","a")
        #ESCRIBIR EN FORMATO CSV
        wanio,wmes,wdia,whora,wmin=escribirFechaCSV(archivo,canal)
        archivo.write("{0:.2f}".format(dato))
        archivo.write(";"+"C"+"\n")
        archivo.close()
        label3C0=label2C0
        label3C0.place(x=0,y=80)
        label2C0=label1C0
        label2C0.place(x=0,y=60)
        label1C0=label0C0
        label1C0.place(x=0,y=40)
        label0C0=Label(canvas10,text=(str(wanio) + "        " +
str(wmes) + "             " + str(wdia)+ "             " +
str(whora)+"             "+str(wmin)+"
"+"{0:.2f}".format(dato)+"            "+"C"),bg="white")
        label0C0.place(x=0,y=20)
        if dato>maxC0:
            enviarAlarma("Se ha sobrepasado la Temperatura MAXIMA de:
"+str(maxC0) + " C")
        if dato<minC0:
            enviarAlarma("Se ha sobrepasado la Temperatura MINIMA de:
"+str(minC0)+" C")

    elif canal ==1:
        archivo = open
("/media/root/ALBERTO/DATALOGGER2_tarjeta/CSV/canal1.csv","a")
        wanio,wmes,wdia,whora,wmin=escribirFechaCSV(archivo,canal)
        archivo.write("{0:.2f}".format(dato)))
        archivo.write(";"+"%"+"\n")
        archivo.close()
        label3C1=label2C1
        label3C1.place(x=0,y=80)
        label2C1=label1C1
        label2C1.place(x=0,y=60)
        label1C1=label0C1
        label1C1.place(x=0,y=40)
        label0C1=Label(canvas20,text=(str(wanio) + "        " +
str(wmes) + "             " + str(wdia)+ "             " +
str(whora)+"             "+str(wmin)+"
"+"{0:.2f}".format(dato))+"            "+"%"),bg="white")
        label0C1.place(x=0,y=20)
        if dato>maxC1:
            enviarAlarma("Se ha sobrepasado la Humedad Relativa
MAXIMA del: "+str(maxC1)+" %")
        if dato<minC1:
            enviarAlarma("Se ha sobrepasado la Humedad Relativa MINIMA
del: "+str(minC1)+" %")
    else:
        archivo = open
("/media/root/ALBERTO/DATALOGGER2_tarjeta/CSV/canal2.csv","a")
        wanio,wmes,wdia,whora,wmin=escribirFechaCSV(archivo,canal)
        archivo.write("{0:.2f}".format(dato))
        archivo.write(";"+"mm"+"\n")
        archivo.close()
        label3C2=label2C2
```

```
            label3C2.place(x=0,y=80)
            label2C2=label1C2
            label2C2.place(x=0,y=60)
            label1C2=label0C2
            label1C2.place(x=0,y=40)
            label0C2=Label(canvas30,text=(str(wanio) + "         " +
    str(wmes) + "              " + str(wdia)+ "              " +
    str(whora)+"             "+str(wmin)+"
    "+"{0:.2f}".format(dato))+"            "+"mm"),bg="white")
            label0C2.place(x=0,y=20)
            if dato>maxC2:
                enviarAlarma("Se ha sobrepasado la Cantidad de Lluvia
    MAXIMA de: "+str(maxC2)+" mm")
            if dato<minC2:
                enviarAlarma("Se ha sobrepasado la Cantidad de Lluvia
    MINIMA de: "+str(minC2)+" mm")
        print (dato)
        print("se han pasado los datos")
        time.sleep(1)

    def escribirFechaCSV(archivo,canal):
        x=datetime.now()
        anioactual=x.year
        mesactual=x.month
        diasemanaactual=time.strftime("%A")
        diaactual=x.day
        horaactual=x.hour
        minactual= x.minute
        archivo.write(str(canal))
        archivo.write(";")
        archivo.write(str(anioactual))
        archivo.write(";")
        archivo.write(str(mesactual))
        archivo.write(";")
        archivo.write(str(diaactual))
        archivo.write(";")
        archivo.write(str(horaactual))
        archivo.write(";")
        archivo.write(str(minactual))
        archivo.write(";")
        return anioactual,mesactual,diaactual,horaactual,minactual
    #Creacion de radiobuttons
    Radiobutton(vConfig, text = "Crear nuevo archivo",
    bg="gold",font=5,variable = selMODIFICAR, value = 0).place(x=10,y=10)
    Radiobutton(vConfig, text = "Modificar archivo
    existente",bg="gold",font=5, variable = selMODIFICAR, value =
    1).place(x=10,y=30)
    #Funcion para guardar las variables establecidas por el usuario
    def Guardar():
        global tPerC0,tPerC1,tPerC2,MODIFICAR
        global horaC0,horaC1,horaC2,minutoC0,minutoC1,minutoC2
        MODIFICAR=selMODIFICAR.get()
        if MODIFICAR==0:
            archivo =
    open("/media/root/ALBERTO/DATALOGGER2_tarjeta/CSV/canal0.csv", "w")
            archivo.write("Canal" + ";" +"Ano" + ";" + "Mes" + ";"+ "Dia"
    + ";" + "Hora" + ";" + "Minuto" + ";" + "Dato" + ";" + "Unidades" +
    "\n")
            archivo.close()
            archivo =
    open("/media/root/ALBERTO/DATALOGGER2_tarjeta/CSV/canal1.csv", "w")
```

```
        archivo.write("Canal" + ";" +"Ano" + ";" + "Mes" + ";"+ "Dia"
+ ";" + "Hora" + ";" + "Minuto" + ";" + "Dato" + ";" + "Unidades" +
"\n")
        archivo.close()
        archivo =
open("/media/root/ALBERTO/DATALOGGER2_tarjeta/CSV/canal2.csv", "w")
        archivo.write("Canal" + ";" +"Ano" + ";" + "Mes" + ";"+ "Dia"
+ ";" + "Hora" + ";" + "Minuto" + ";" + "Dato" + ";" + "Unidades" +
"\n")
        archivo.close()
    tPerC0=convertirTiempo(tPeriodoC0)
    tPerC1=convertirTiempoC1(tPeriodoC1)
    tPerC2=convertirTiempoC2(tPeriodoC2)
    print("canal 0", horaC0,minutoC0,"   tPeriodo",tPerC0)
    print("canal 1", horaC1,minutoC1,"   tPeriodo",tPerC1)
    print("canal 2", horaC2,minutoC2,"   tPeriodo",tPerC2)
#Obtener la fecha establecida
def convertirDias():
    lC0=selLunesC0.get()
    mC0=selMartesC0.get()
    xC0=selMiercolesC0.get()
    jC0=selJuevesC0.get()
    vC0=selViernesC0.get()
    sC0=selSabadoC0.get()
    dC0=selDomingoC0.get()
    diasC0=[lC0,mC0,xC0,jC0,vC0,sC0,dC0]
    lC1=selLunesC1.get()
    mC1=selMartesC1.get()
    xC1=selMiercolesC1.get()
    jC1=selJuevesC1.get()
    vC1=selViernesC1.get()
    sC1=selSabadoC1.get()
    dC1=selDomingoC1.get()
    diasC1=[lC1,mC1,xC1,jC1,vC1,sC1,dC1]
    lC2=selLunesC2.get()
    mC2=selMartesC2.get()
    xC2=selMiercolesC2.get()
    jC2=selJuevesC2.get()
    vC2=selViernesC2.get()
    sC2=selSabadoC2.get()
    dC2=selDomingoC2.get()
    diasC2=[lC2,mC2,xC2,jC2,vC2,sC2,dC2]
    print diasC0
    return diasC0,diasC1,diasC2
#Creacion de botones
bSTART=Button(ventana,text="START",bg="green",fg="white",font=("Times"
, 13, "bold"),command=ContarTiempo)
bSTART.place(x=50,y=50)
bSTOP=Button(ventana,text="STOP",bg="red",fg="white",font=("Times",
13, "bold"),command=STOP)
bSTOP.place(x=50,y=90)
bCONFIG=Button(ventana,text="CONFIGURACION",bg="yellow",fg="black",fon
t=("Times", 13, "bold"),command=abrirConfig)
bCONFIG.place(x=50,y=130)
bSalir=Button(vConfig,text="SALIR",bg="orange",fg="white",font=("Times
", 12, "bold"),command=Salir)
bSalir.place(x=150,y=65)
bGuardar=Button(vConfig,text="GUARDAR",bg="green",fg="white",font=("Ti
mes", 12, "bold"), command=Guardar)
bGuardar.place(x=10,y=65)
bMOSTRAR=Button(ventana,text="Mostrar Datos\n\/",bg="dark
```

```
turquoise",fg="white",font=("Times", 9, "bold"),command=MOSTRAR)
bMOSTRAR.place(x=50,y=200)
bOCULTAR=Button(ventana,text="/\ \nOcultar Datos
",bg="pink",fg="black",font=("Times", 9, "bold"),command=OCULTAR)
bOCULTAR.place_forget()

#CANAL0
Radiobutton(vConfig, text = "Captura por Fecha", font=5,bg="red",
variable = selprogPeriodo0, value = 0).place(x=290,y=120)
Radiobutton(vConfig, text = "Captura periodica",font=5,bg="red",
variable = selprogPeriodo0, value = 1).place(x=590,y=120)
#hora
Label(vConfig,text="Configure
hora",font=5,bg="white").place(x=400,y=150)
Label(vConfig,text=str(horaC0),bg="white",font=5,padx=10).place(x=450,
y=180)
def restaHoraC0():
    global horaC0
    if (horaC0>0):
        horaC0=horaC0-1

Label(vConfig,text=str(horaC0),bg="white",font=5,padx=10).place(x=450,
y=180)
    else:
        horaC0=23

Label(vConfig,text=str(horaC0),bg="white",font=5,padx=10).place(x=450,
y=180)
def sumaHoraC0():
    global horaC0
    if (horaC0<23):
        horaC0=horaC0+1

Label(vConfig,text=str(horaC0),bg="white",font=5,padx=10).place(x=450,
y=180)
    else:
        horaC0=0

Label(vConfig,text=str(horaC0),bg="white",font=5,padx=10).place(x=450,
y=180)
Button(vConfig,text="<",font=("Times", 8,
"bold"),command=restaHoraC0).place(x=400,y=180)
Button(vConfig,text=">",font=("Times", 8,
"bold"),command=sumaHoraC0).place(x=500,y=180)
#minuto
Label(vConfig,text="Configure
minuto",bg="white",font=5).place(x=400,y=220)
Label(vConfig,text=str(minutoC0),bg="white",font=5,padx=8).place(x=450
,y=250)
def restaMinC0():
    global minutoC0
    if (minutoC0>0):
        minutoC0=minutoC0-1

Label(vConfig,text=str(minutoC0),bg="white",font=5,padx=8).place(x=450
,y=250)
    else:
        minutoC0=59

Label(vConfig,text=str(minutoC0),bg="white",font=5,padx=8).place(x=450
,y=250)
```

```
def sumaMinC0():
    global minutoC0
    if (minutoC0<59):
        minutoC0=minutoC0+1

Label(vConfig,text=str(minutoC0),bg="white",font=5,padx=8).place(x=450
,y=250)
    else:
        minutoC0=0

Label(vConfig,text=str(minutoC0),bg="white",font=5,padx=8).place(x=450
,y=250)
Button(vConfig,text="<",font=("Times", 8,
"bold"),command=restaMinC0).place(x=400,y=250)
Button(vConfig,text=">",font=("Times", 8,
"bold"),command=sumaMinC0).place(x=500,y=250)
#dia
selLunesC0=StringVar()
selMartesC0=StringVar()
selMiercolesC0=StringVar()
selJuevesC0=StringVar()
selViernesC0=StringVar()
selSabadoC0=StringVar()
selDomingoC0=StringVar()
Checkbutton(vConfig,text="Lunes",variable=selLunesC0,onvalue='Monday',
state=NORMAL).place(x=300,y=150)
Checkbutton(vConfig,text="Martes",variable=selMartesC0,onvalue='Tuesda
y',state=NORMAL).place(x=300,y=170)
Checkbutton(vConfig,text="Miercoles",variable=selMiercolesC0,onvalue='
Wednesday',state=NORMAL).place(x=300,y=190)
Checkbutton(vConfig,text="Jueves",variable=selJuevesC0,onvalue='Thursd
ay',state=NORMAL).place(x=300,y=210)
Checkbutton(vConfig,text="Viernes",variable=selViernesC0,onvalue='Frid
ay',state=NORMAL).place(x=300,y=230)
Checkbutton(vConfig,text="Sabado",variable=selSabadoC0,onvalue='Saturd
ay',state=NORMAL).place(x=300,y=250)
Checkbutton(vConfig,text="Domingo",variable=selDomingoC0,onvalue='Sund
ay',state=NORMAL).place(x=300,y=270)
#periodico
def restaPeriodoC0():
    global tPeriodoC0
    if (tPeriodoC0>0):
        tPeriodoC0=tPeriodoC0-1

Label(vConfig,text=str(tPeriodoC0),bg="white",font=5,padx=8).place(x=6
50,y=200)
def sumaPeriodoC0():
    global tPeriodoC0
    if (tPeriodoC0<100):
        tPeriodoC0=tPeriodoC0+1

Label(vConfig,text=str(tPeriodoC0),bg="white",font=5,padx=8).place(x=6
50,y=200)
Label(vConfig,text="Captura cada:
",bg="white",font=5).place(x=600,y=170)
Label(vConfig,text=str(tPeriodoC0),bg="white",font=5,padx=8).place(x=6
50,y=200)
Button(vConfig,text="<",font=("Times", 8,
"bold"),command=restaPeriodoC0).place(x=600,y=200)
Button(vConfig,text=">",font=("Times", 8,
"bold"),command=sumaPeriodoC0).place(x=700,y=200)
```

```
seluTiempoC0=IntVar()
Radiobutton(vConfig, text = "minutos", variable = seluTiempoC0, value
= 0).place(x=750,y=220)
Radiobutton(vConfig, text = "horas", variable = seluTiempoC0, value =
1).place(x=750,y=240)
Radiobutton(vConfig, text = "dias", variable = seluTiempoC0, value =
2).place(x=750,y=260)
Radiobutton(vConfig, text = "segundos", variable = seluTiempoC0, value
= 3).place(x=750,y=200)
def convertirTiempo(tiempo):
    uTiempoC0=seluTiempoC0.get()
    print (uTiempoC0)
    if uTiempoC0==0:
        tiempo_conv=tiempo*60
    elif uTiempoC0==1:
        tiempo_conv=tiempo*3600
    elif uTiempoC0==2:
        tiempo_conv=tiempo*3600*24
    else:
        tiempo_conv=tiempo*1
    return tiempo_conv
############################################################
#CANAL 1
Radiobutton(vConfig, text = "Captura por Fecha",
bg="red",font=5,variable = selprogPeriodo1, value =
0).place(x=290,y=320)
Radiobutton(vConfig, text = "Captura periodica",
bg="red",font=5,variable = selprogPeriodo1, value =
1).place(x=590,y=320)
#hora
Label(vConfig,text="Configure
hora",bg="white",font=5).place(x=400,y=350)
Label(vConfig,text=str(horaC1),bg="white",font=5,padx=8).place(x=450,y
=380)

def restaHoraC1():
    global horaC1
    if (horaC1>0):
        horaC1=horaC1-1

Label(vConfig,text=str(horaC1),bg="white",font=5,padx=8).place(x=450,y
=380)
    else:
        horaC1=23

Label(vConfig,text=str(horaC1),bg="white",font=5,padx=8).place(x=450,y
=380)
def sumaHoraC1():
    global horaC1
    if (horaC1<23):
        horaC1=horaC1+1

Label(vConfig,text=str(horaC1),bg="white",font=5,padx=8).place(x=450,y
=380)
    else:
        horaC1=0

Label(vConfig,text=str(horaC1),bg="white",font=5,padx=8).place(x=450,y
=380)
```

```
Button(vConfig,text="<",font=("Times", 8,
"bold"),command=restaHoraC1).place(x=400,y=380)
Button(vConfig,text=">",font=("Times", 8,
"bold"),command=sumaHoraC1).place(x=500,y=380)

#minuto
Label(vConfig,text="Configure
minuto",bg="white",font=5).place(x=400,y=420)
Label(vConfig,text=str(minutoC1),bg="white",font=5,padx=8).place(x=450
,y=450)
def restaMinC1():
    global minutoC1
    if (minutoC1>0):
        minutoC1=minutoC1-1

Label(vConfig,text=str(minutoC1),bg="white",font=5,padx=8).place(x=450
,y=450)
    else:
        minutoC1=59

Label(vConfig,text=str(minutoC1),bg="white",font=5,padx=8).place(x=450
,y=450)
def sumaMinC1():
    global minutoC1
    if (minutoC1<59):
        minutoC1=minutoC1+1

Label(vConfig,text=str(minutoC1),bg="white",font=5,padx=8).place(x=450
,y=450)
    else:
        minutoC1=0

Label(vConfig,text=str(minutoC1),bg="white",font=5,padx=8).place(x=450
,y=450)

Button(vConfig,text="<",font=("Times", 8,
"bold"),command=restaMinC1).place(x=400,y=450)
Button(vConfig,text=">",font=("Times", 8,
"bold"),command=sumaMinC1).place(x=500,y=450)
#dia
selLunesC1=StringVar()
selMartesC1=StringVar()
selMiercolesC1=StringVar()
selJuevesC1=StringVar()
selViernesC1=StringVar()
selSabadoC1=StringVar()
selDomingoC1=StringVar()
Checkbutton(vConfig,text="Lunes",variable=selLunesC1,onvalue='Monday')
.place(x=300,y=350)
Checkbutton(vConfig,text="Martes",variable=selMartesC1,onvalue='Tuesda
y').place(x=300,y=370)
Checkbutton(vConfig,text="Miercoles",variable=selMiercolesC1,onvalue='
Wednesday').place(x=300,y=390)
Checkbutton(vConfig,text="Jueves",variable=selJuevesC1,onvalue='Thursd
ay').place(x=300,y=410)
Checkbutton(vConfig,text="Viernes",variable=selViernesC1,onvalue='Frid
ay').place(x=300,y=430)
Checkbutton(vConfig,text="Sabado",variable=selSabadoC1,onvalue='Saturd
ay').place(x=300,y=450)
Checkbutton(vConfig,text="Domingo",variable=selDomingoC1,onvalue='Sund
ay').place(x=300,y=470)
```

```
#periodico
def restaPeriodoC1():
    global tPeriodoC1
    if (tPeriodoC1>0):
        tPeriodoC1=tPeriodoC1-1

Label(vConfig,text=str(tPeriodoC1),bg="white",font=5,padx=8).place(x=6
50,y=400)
def sumaPeriodoC1():
    global tPeriodoC1
    if (tPeriodoC1<100):
        tPeriodoC1=tPeriodoC1+1

Label(vConfig,text=str(tPeriodoC1),bg="white",font=5,padx=8).place(x=6
50,y=400)
Label(vConfig,text="Captura cada:
",bg="white",font=5).place(x=600,y=370)
Label(vConfig,text=str(tPeriodoC1),bg="white",font=5,padx=8).place(x=6
50,y=400)
Button(vConfig,text="<",font=("Times", 8,
"bold"),command=restaPeriodoC1).place(x=600,y=400)
Button(vConfig,text=">",font=("Times", 8,
"bold"),command=sumaPeriodoC1).place(x=700,y=400)

seluTiempoC1=IntVar()
Radiobutton(vConfig, text = "minutos", variable = seluTiempoC1, value
= 0).place(x=750,y=420)
Radiobutton(vConfig, text = "horas", variable = seluTiempoC1, value =
1).place(x=750,y=440)
Radiobutton(vConfig, text = "dias", variable = seluTiempoC1, value =
2).place(x=750,y=460)
Radiobutton(vConfig, text = "segundos", variable = seluTiempoC1, value
= 3).place(x=750,y=400)

def convertirTiempoC1(tiempo):
    uTiempoC1=seluTiempoC1.get()
    print (uTiempoC1)
    if uTiempoC1==0:
        tiempo_conv=tiempo*60
    elif uTiempoC1==1:
        tiempo_conv=tiempo*3600
    elif uTiempoC1==2:
        tiempo_conv=tiempo*3600*24
    else:
        tiempo_conv=tiempo*1
    return tiempo_conv
########################################################################
######################
#CANAL 2
Radiobutton(vConfig, text = "Captura por Fecha",
bg="red",font=5,variable = selprogPeriodo2, value =
0).place(x=290,y=520)
Radiobutton(vConfig, text = "Captura periodica",
bg="red",font=5,variable = selprogPeriodo2, value =
1).place(x=590,y=520)
#hora
Label(vConfig,text="Configure
hora",bg="white",font=5).place(x=400,y=550)
Label(vConfig,text=str(horaC2),bg="white",font=5,padx=8).place(x=450,y
=580)
```

```
def restaHoraC2():
    global horaC2
    if (horaC2>0):
        horaC2=horaC2-1

Label(vConfig,text=str(horaC2),bg="white",font=5,padx=8).place(x=450,y
=580)
    else:
        horaC2=23

Label(vConfig,text=str(horaC2),bg="white",font=5,padx=8).place(x=450,y
=580)
def sumaHoraC2():
    global horaC2
    if (horaC2<23):
        horaC2=horaC2+1

Label(vConfig,text=str(horaC2),bg="white",font=5,padx=8).place(x=450,y
=580)
    else:
        horaC2=0

Label(vConfig,text=str(horaC2),bg="white",font=5,padx=8).place(x=450,y
=580)

Button(vConfig,text="<",font=("Times", 8,
"bold"),command=restaHoraC2).place(x=400,y=580)
Button(vConfig,text=">",font=("Times", 8,
"bold"),command=sumaHoraC2).place(x=500,y=580)

#minuto
Label(vConfig,text="Configure
minuto",bg="white",font=5).place(x=400,y=620)
Label(vConfig,text=str(minutoC2),bg="white",font=5,padx=8).place(x=450
,y=650)
def restaMinC2():
    global minutoC2
    if (minutoC2>0):
        minutoC2=minutoC2-1

Label(vConfig,text=str(minutoC2),bg="white",font=5,padx=8).place(x=450
,y=650)
    else:
        minutoC2=59

Label(vConfig,text=str(minutoC2),bg="white",font=5,padx=8).place(x=450
,y=650)
def sumaMinC2():
    global minutoC2
    if (minutoC2<59):
        minutoC2=minutoC2+1

Label(vConfig,text=str(minutoC2),bg="white",font=5,padx=8).place(x=450
,y=650)
    else:
        minutoC2=0

Label(vConfig,text=str(minutoC2),bg="white",font=5,padx=8).place(x=450
,y=650)

Button(vConfig,text="<",font=("Times", 8,
```

```
"bold"),command=restaMinC2).place(x=400,y=650)
Button(vConfig,text=">",font=("Times", 8,
"bold"),command=sumaMinC2).place(x=500,y=650)
#dia
selLunesC2=StringVar()
selMartesC2=StringVar()
selMiercolesC2=StringVar()
selJuevesC2=StringVar()
selViernesC2=StringVar()
selSabadoC2=StringVar()
selDomingoC2=StringVar()
Checkbutton(vConfig,text="Lunes",variable=selLunesC2,onvalue='Monday',
state=NORMAL).place(x=300,y=550)
Checkbutton(vConfig,text="Martes",variable=selMartesC2,onvalue='Tuesda
y',state=NORMAL).place(x=300,y=570)
Checkbutton(vConfig,text="Miercoles",variable=selMiercolesC2,onvalue='
Wednesday',state=NORMAL).place(x=300,y=590)
Checkbutton(vConfig,text="Jueves",variable=selJuevesC2,onvalue='Thursd
ay',state=NORMAL).place(x=300,y=610)
Checkbutton(vConfig,text="Viernes",variable=selViernesC2,onvalue='Frid
ay',state=NORMAL).place(x=300,y=630)
Checkbutton(vConfig,text="Sabado",variable=selSabadoC2,onvalue='Saturd
ay',state=NORMAL).place(x=300,y=650)
Checkbutton(vConfig,text="Domingo",variable=selDomingoC2,onvalue='Sund
ay',state=NORMAL).place(x=300,y=670)
#periodico
def restaPeriodoC2():
    global tPeriodoC2
    if (tPeriodoC2>0):
        tPeriodoC2=tPeriodoC2-1

Label(vConfig,text=str(tPeriodoC2),bg="white",font=5,padx=8).place(x=6
50,y=600)
def sumaPeriodoC2():
    global tPeriodoC2
    if (tPeriodoC2<100):
        tPeriodoC2=tPeriodoC2+1

Label(vConfig,text=str(tPeriodoC2),bg="white",font=5,padx=8).place(x=6
50,y=600)
Label(vConfig,text="Captura cada:
",bg="white",font=5).place(x=600,y=570)
Label(vConfig,text=str(tPeriodoC2),bg="white",font=5,padx=8).place(x=6
50,y=600)
Button(vConfig,text="<",font=("Times", 8,
"bold"),command=restaPeriodoC2).place(x=600,y=600)
Button(vConfig,text=">",font=("Times", 8,
"bold"),command=sumaPeriodoC2).place(x=700,y=600)

seluTiempoC2=IntVar()
Radiobutton(vConfig, text = "minutos", variable = seluTiempoC2, value
= 0).place(x=750,y=620)
Radiobutton(vConfig, text = "horas", variable = seluTiempoC2, value =
1).place(x=750,y=640)
Radiobutton(vConfig, text = "dias", variable = seluTiempoC2, value =
2).place(x=750,y=660)
Radiobutton(vConfig, text = "segundos", variable = seluTiempoC2, value
= 3).place(x=750,y=600)

def convertirTiempoC2(tiempo):
    uTiempoC2=seluTiempoC2.get()
```

```
    print (uTiempoC2)
    if uTiempoC2==0:
        tiempo_conv=tiempo*60
    elif uTiempoC2==1:
        tiempo_conv=tiempo*3600
    elif uTiempoC2==2:
        tiempo_conv=tiempo*3600*24
    else:
        tiempo_conv=tiempo*1
    return tiempo_conv

#ENTRADA DEL EMAIL
enEmail=Entry(ventana, textvariable=selEmail, width=25)
enEmail.place(x=280,y=120)
enEmail.insert(0, email)
Label(ventana,text="Copia de seguridad a:
",bg="black",fg="white").place(x=280,y=100)
Label(ventana,text="cada: ",bg="black",fg="white").place(x=280,y=140)
enCopia=Entry(ventana, textvariable=selCopia, width=5)
enCopia.place(x=280,y=160)
enCopia.insert(1,str(copia))

seluCopia=IntVar()
Radiobutton(ventana, text =
"horas",background="black",fg="white",selectcolor="red",variable =
seluCopia, value = 0).place(x=350,y=160)
Radiobutton(ventana, text = "dias",
background="black",fg="white",selectcolor="red",variable = seluCopia,
value = 1).place(x=350,y=180)
Radiobutton(ventana, text =
"semanas",background="black",fg="white",selectcolor="red", variable =
seluCopia, value = 2).place(x=350,y=200)
Radiobutton(ventana, text = "meses",
background="black",fg="white",selectcolor="red",variable = seluCopia,
value = 3).place(x=350,y=220)

def convertirTiempoCopia(tiempo):
    uCopia=seluCopia.get()
    print (uCopia)
    if uCopia==0:
        tiempo_conv=tiempo*3600
    elif uCopia==1:
        tiempo_conv=tiempo*3600*24
    elif uCopia==2:
        tiempo_conv=tiempo*3600*24*7
    else:
        tiempo_conv=tiempo*3600*24*30
    return tiempo_conv

selmaxC0=IntVar()
selminC0=IntVar()
selmaxC1=IntVar()
selminC1=IntVar()
selmaxC2=IntVar()
selminC2=IntVar()
maxC0=50
minC0=0
maxC1=90
minC1=10
maxC2=60
minC2=5
```

```
Label(vConfig,text="Canal TEMPERATURA",bg="lawn
green",fg="black",font=("Times", 18,
"bold"),anchor=CENTER).place(x=30,y=120)
Label(vConfig,text="Canal HUMEDAD          ",bg="lawn
green",fg="black",font=("Times", 18,
"bold"),anchor=CENTER).place(x=30,y=320)
Label(vConfig,text="Canal LLUVIA               ",bg="lawn
green",fg="black",font=("Times", 18,
"bold"),anchor=CENTER).place(x=30,y=520)

Label(vConfig,text="AVISO A
\nTEMPERATURA",bg="red",fg="white",font=("Times", 12,
"bold"),anchor=CENTER).place(x=30,y=170)
Label(vConfig,text="AVISO A
\nHUMEDAD",bg="red",fg="white",font=("Times", 12,
"bold"),anchor=CENTER, padx=20).place(x=30,y=370)
Label(vConfig,text="AVISO A
\nLLUVIA",bg="red",fg="white",font=("Times", 12,
"bold"),anchor=CENTER, padx=25).place(x=30,y=570)

def convertirMaxMin():
    global maxC0,minC0,maxC1,minC1,maxC2,minC2
    maxC0=selmaxC0.get()
    minC0=selminC0.get()
    maxC1=selmaxC1.get()
    minC1=selminC1.get()
    maxC2=selmaxC2.get()
    minC2=selminC2.get()

enMaxC0=Entry(vConfig, textvariable=selmaxC0, width=5)
enMaxC0.place(x=130,y=230)
enMaxC0.insert(1,str(maxC0))
Label(vConfig,text="MAXIMA ( C)",bg="cyan",fg="black",font=("Times",
10, "bold")).place(x=30,y=230)
enMinC0=Entry(vConfig, textvariable=selminC0, width=5)
enMinC0.place(x=130,y=260)
enMinC0.insert(1,str(minC0))
Label(vConfig,text="MINIMA ( C)",bg="cyan",fg="black",font=("Times",
10, "bold")).place(x=30,y=260)

enMaxC1=Entry(vConfig, textvariable=selmaxC1, width=5)
enMaxC1.place(x=130,y=430)
enMaxC1.insert(1,str(maxC1))
Label(vConfig,text="MAXIMA (%)",bg="cyan",fg="black",font=("Times",
10, "bold")).place(x=30,y=430)
enMinC0=Entry(vConfig, textvariable=selminC1, width=5)
enMinC0.place(x=130,y=460)
enMinC0.insert(1,str(minC1))
Label(vConfig,text="MINIMA (%)",bg="cyan",fg="black",font=("Times",
10, "bold")).place(x=30,y=460)

enMaxC2=Entry(vConfig, textvariable=selmaxC2, width=5)
enMaxC2.place(x=130,y=630)
enMaxC2.insert(1,str(maxC2))
Label(vConfig,text="MAXIMA (mm)",bg="cyan",fg="black",font=("Times",
10, "bold")).place(x=30,y=630)
enMinC2=Entry(vConfig, textvariable=selminC2, width=5)
enMinC2.place(x=130,y=660)
enMinC2.insert(1,str(minC2))
Label(vConfig,text="MINIMA (mm)",bg="cyan",fg="black",font=("Times",
```

```
10, "bold")).place(x=30,y=660)
ventana.mainloop()
```

3. MANUAL DE USUARIO

3.1. Conexiones externas

Con el Switch SW8 colocado en la posición OFF, como se indica en la siguiente Figura 1, se debe conectar la tarjeta ZedBoard a los siguientes dispositivos externos:

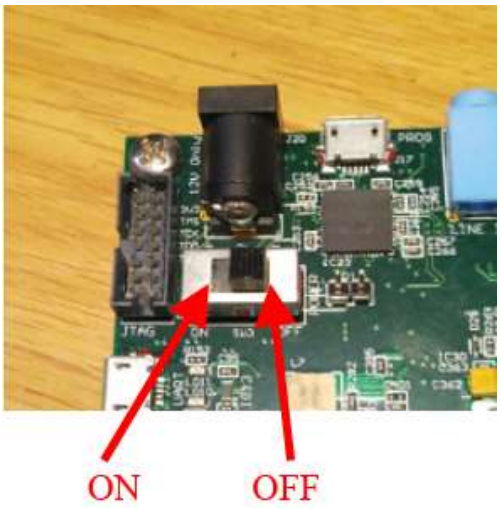


Figura 1: Switch de encendido/apagado SW8

ZedBoard	**Conexión externa**
USB	Hub de USB (teclado, ratón, pendrive)
Power 12V	Tensión de red
Ethernet	Salida de un router con internet
VGA	Monitor
ADC	Entradas analógicas de tensión a muestrear
SD	Insertar tarjeta SD

Tabla 1: Conexiones externas de la ZedBoard

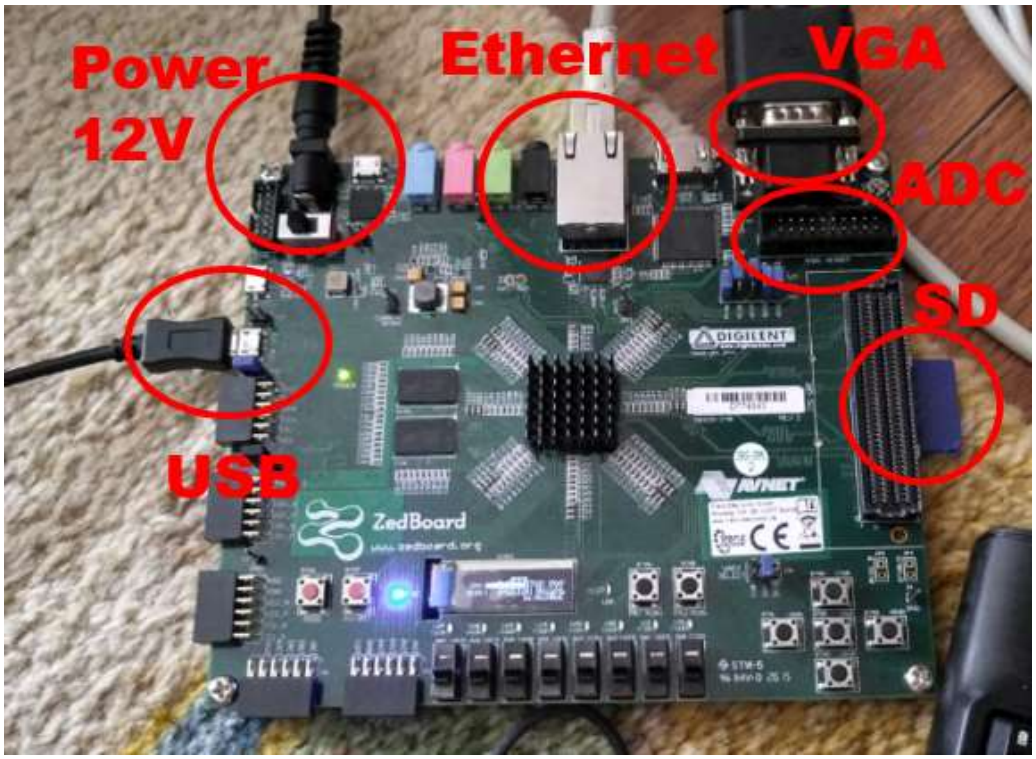


Figura 2: Salidas externas de la ZedBoard

Además, hay que conectar al XADC el circuito de simulación externo, cada canal con su correspondiente pin del XADC, como se indica en la siguiente Figura 3.

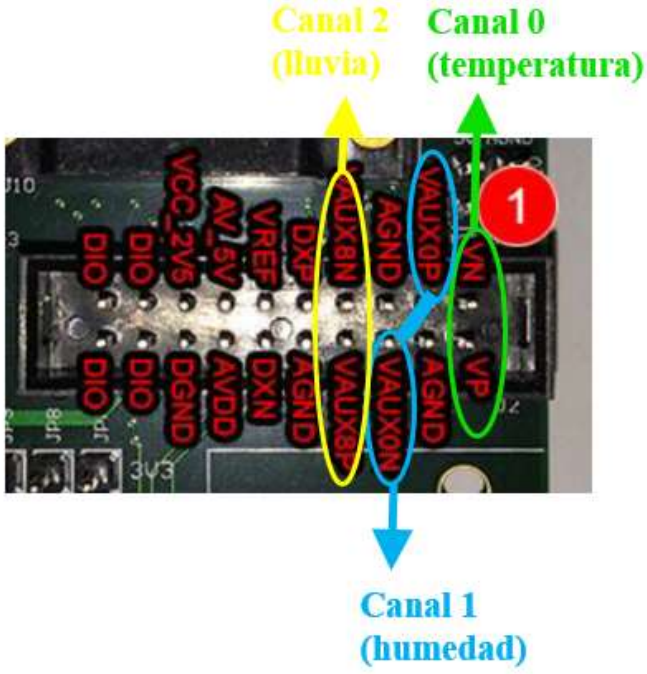


Figura 3: Conexiones del circuito con el XADC.

3.2. Colocación de los jumpers

Hay que provocar que Zynq arranque leyendo el contenido de la tarjeta SD. Para ello, se deben colocar los jumpers que se encuentran situados en la Figura 4, de la siguiente manera:

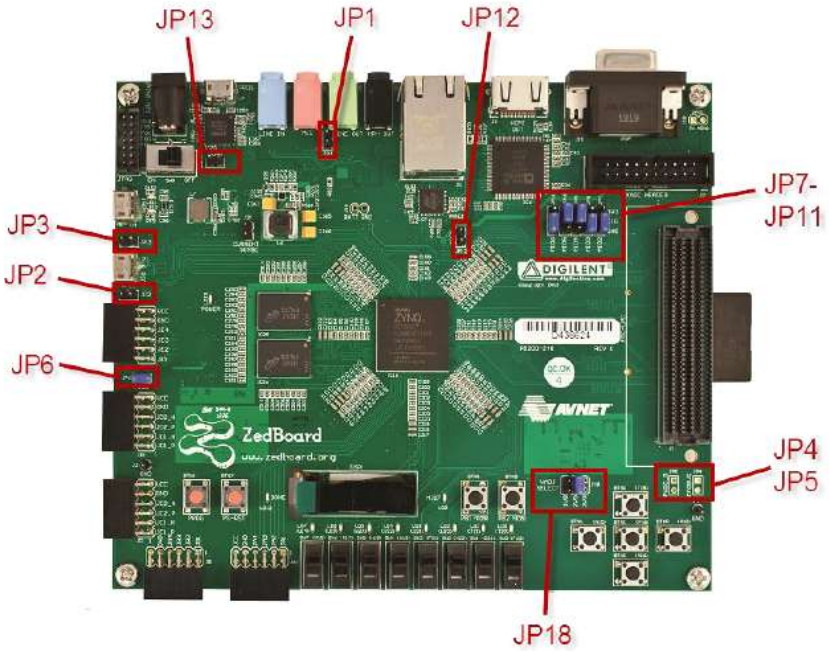


Figura 4: Situación de los Jumpers en la ZedBoard

JP11	0
JP10	1
JP9	1
JP8	0
JP7	0
JP6	1

Tabla 2: Colocación de los Jumpers de arranque

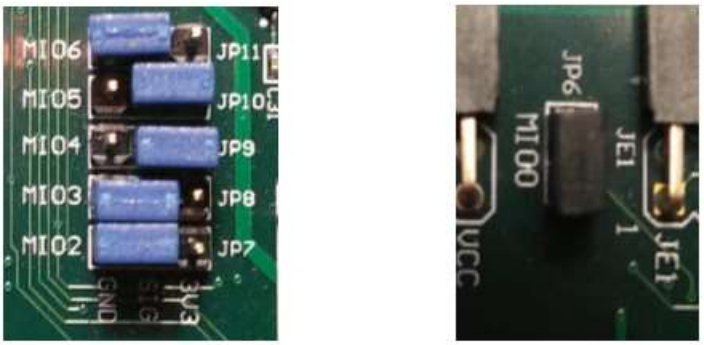


Figura 5: Colocación de los Jumpers de arranque

3.3. Interfaz de la aplicación

Tras realizar las conexiones iniciales, se debe poner a ON el Switch SW8 de la Figura 6. Tardará unos segundos en arrancar el sistema y cuando acabe se podrá ver por pantalla una terminal para poder escribir. Para poder ver la interfaz gráfica se debe escribir el comando:

```
startx
```

Aparecerá un fondo de escritorio como el de un ordenador. Ahí se encuentra el archivo DATALOGGER2.py, en el que al pinchar sobre él nos dará la opción de ejecutar *Execute* (Figura 6)

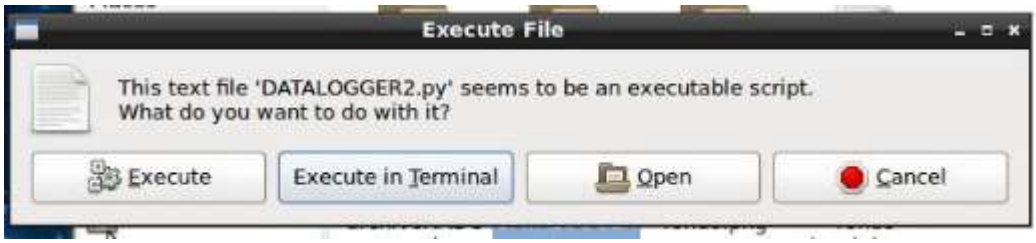


Figura 6: Ejecutar el archivo DATALOGGER.py

La interfaz que muestra la aplicación y la explicación del funcionamiento de sus botones es la que se muestra en la Figura 7.

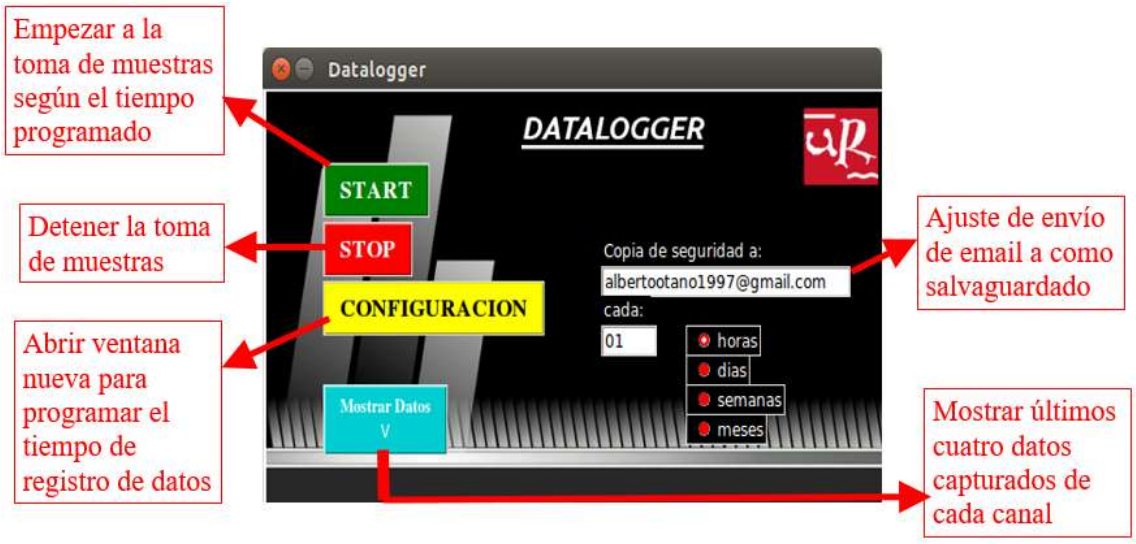


Figura 7: Interfaz con el usuario y funcionamiento de sus botones

El primer paso es programar el tiempo que el usuario desea que se tomen muestras de datos. Para ello debe pulsar el botón CONFIGURACION y se abrirá una ventana como la de la Figura 8.

Figura 8: Menú configuración

Esta opción permite programar cada uno de los tres canales de registro de datos. Existen dos opciones de programación:

- **Captura por Fecha**: permite ajustar los días, la hora y los minutos que se quiere que se registren datos. Por ejemplo: se pretenden registrar los datos los lunes, martes y miércoles a las 18:30
- **Captura Periódica**: con esta opción se puede programar que el sistema registre datos cada un cierto tiempo. Por ejemplo: se registren los datos los datos cada 30 minutos.

En esta pantalla también se pueden ajustar más opciones como:

- **Crear nuevo archivo**: esta opción sobrescribe los archivos CSV existentes en cada canal. Es decir, se borrarán los datos previos que se han registrado hasta la fecha.
- **Modificar archivo existente**: mediante esta opción se pueden mantener los datos previos. Es decir, los datos que se registren a partir de este momento se añadirán a los ya existentes.

La aplicación también permite enviar un email de alarma en caso de que se sobrepase los umbrales máximo o mínimo de cada canal. Estos umbrales han de ser establecidos por el usuario en el Menú Configuración (Figura 8). Un ejemplo del mail de alarma que se envía se muestra en la Figura 9.

Figura 9: Email de alarma por sobre pasamiento de temperatura.

Por último, el usuario deberá pulsar el botón de GUARDAR, para que se registren las fechas y tiempos establecidos. Si ya no desea seguir configurando tiempos, debe pulsar el botón SALIR.

Para empezar el registro de datos, el usuario debe pulsar el botón START. Si se desea visualizar los datos que se han ido registrando, existe el botón Mostrar Datos, que mostrara los últimos cuatro datos registrados de cada canal, abriendo un desplegable como el de la Figura 10.

En el momento en que el usuario desee parar el registro de datos, deberá pulsar el botón STOP.

Figura 10: Menú de visualización de datos registrados.

Los datos que se vayan registrando se guardaran en el dispositivo USB, en el directorio: /DATALOGGER2_tarjeta/CSV. En esa carpeta se encontrarán tres archivos con la extensión .csv, cada uno correspondiente a un canal (temperatura, humedad relativa y lluvia).

Por último, esta aplicación permite enviar dichos archivos CSV al correo electrónico a modo de salvaguardado. El usuario previamente ha establecido el correo al que se mandaran los archivos y cada cuanto tiempo en el menú principal (Figura 11)

- Asunto: “Datos hasta la fecha: <fecha y hora del salvaguardado>
- Remitente: tfgalbertootano@gmail.com
- Destinatario: <Gmail establecido por el usuario>
- Adjunto: canalTemperatura.csv, canalHumedad.csv, canalLluvia.csv

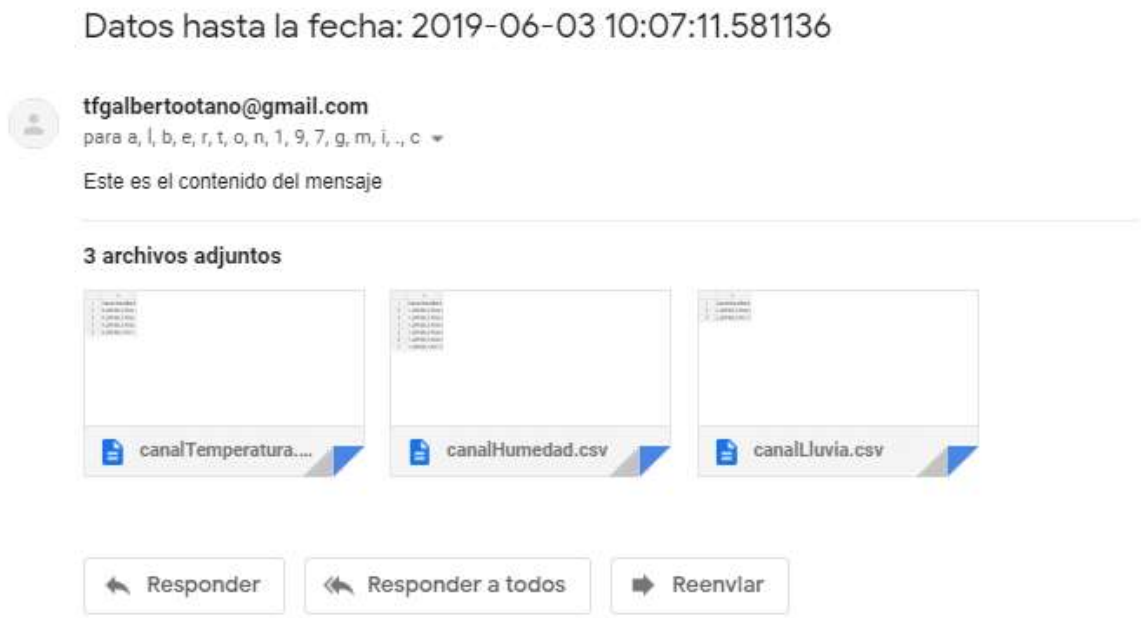


Figura 11: Email enviado por la aplicación

En caso de utilizar un dispositivo USB externo para el almacenamiento de los datos, es necesario formatear dicho dispositivo con el nombre “ALBERTO” y crear un directorio para el almacenamiento de datos dentro del USB con la ruta /DATALOGGER2_tarjeta/CSV/

UNIVERSIDAD DE LA RIOJA

Trabajo Fin de Grado

Desarrollo de un “Datalogger” basado en ARM Cortex A9 y Linux mediante dispositivo Zynq-7000

4. PLANOS

Alberto Otaño Jiménez

Julio 2019

PLANOS

1. Entrada al conversor XADC

2. Esquema de las conexiones del XADC con la FIFO con Xillybus

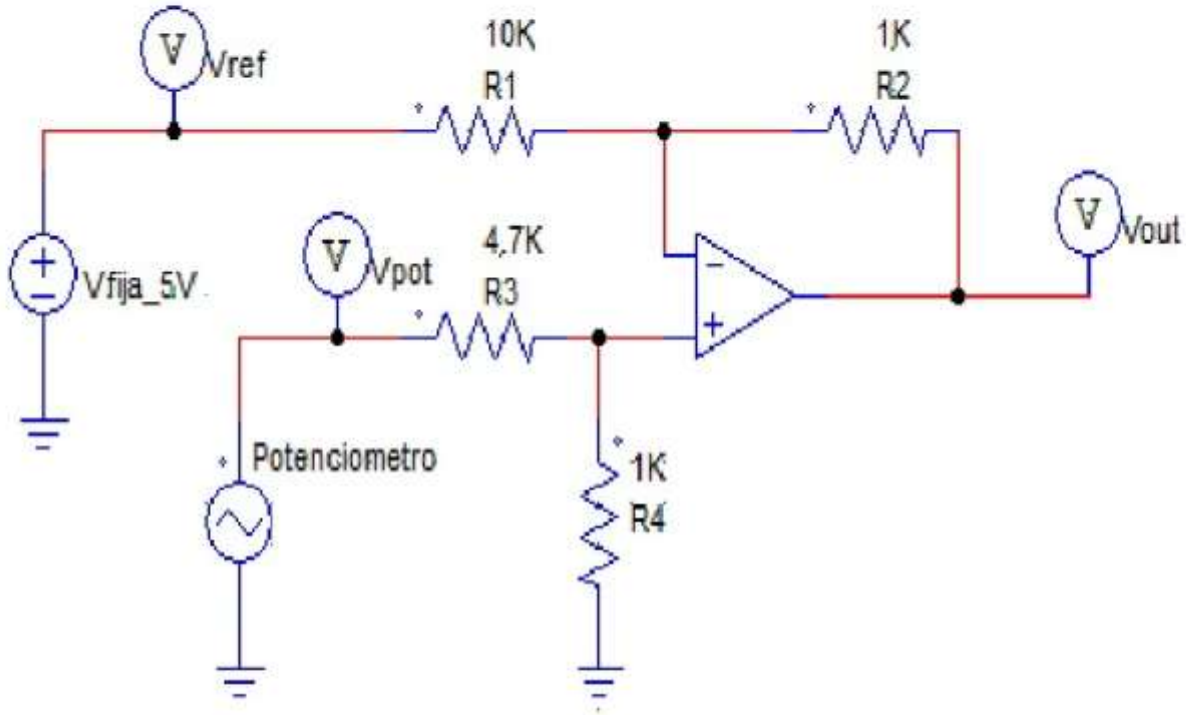
3. Diseño esquemático de la FPGA

Los planos del diseño esquemático de la tarjeta ZedBoard pertenecen a la empresa Xilinx y pueden ser consulados en el siguiente enlace: [3] http://zedboard.com/misc/files/ZedBoard_RevC.1_Schematic_preliminary.pdf

4. Diseño mecánico de la FPGA

Los planos del diseño mecánico de la tarjeta ZedBoard pertenecen a la empresa Xilinx y pueden ser consulados en el siguiente enlace: [4] http://zedboard.com/misc/files/ZedBoard_RevC.1_MechDrawing_preliminary.pdf y

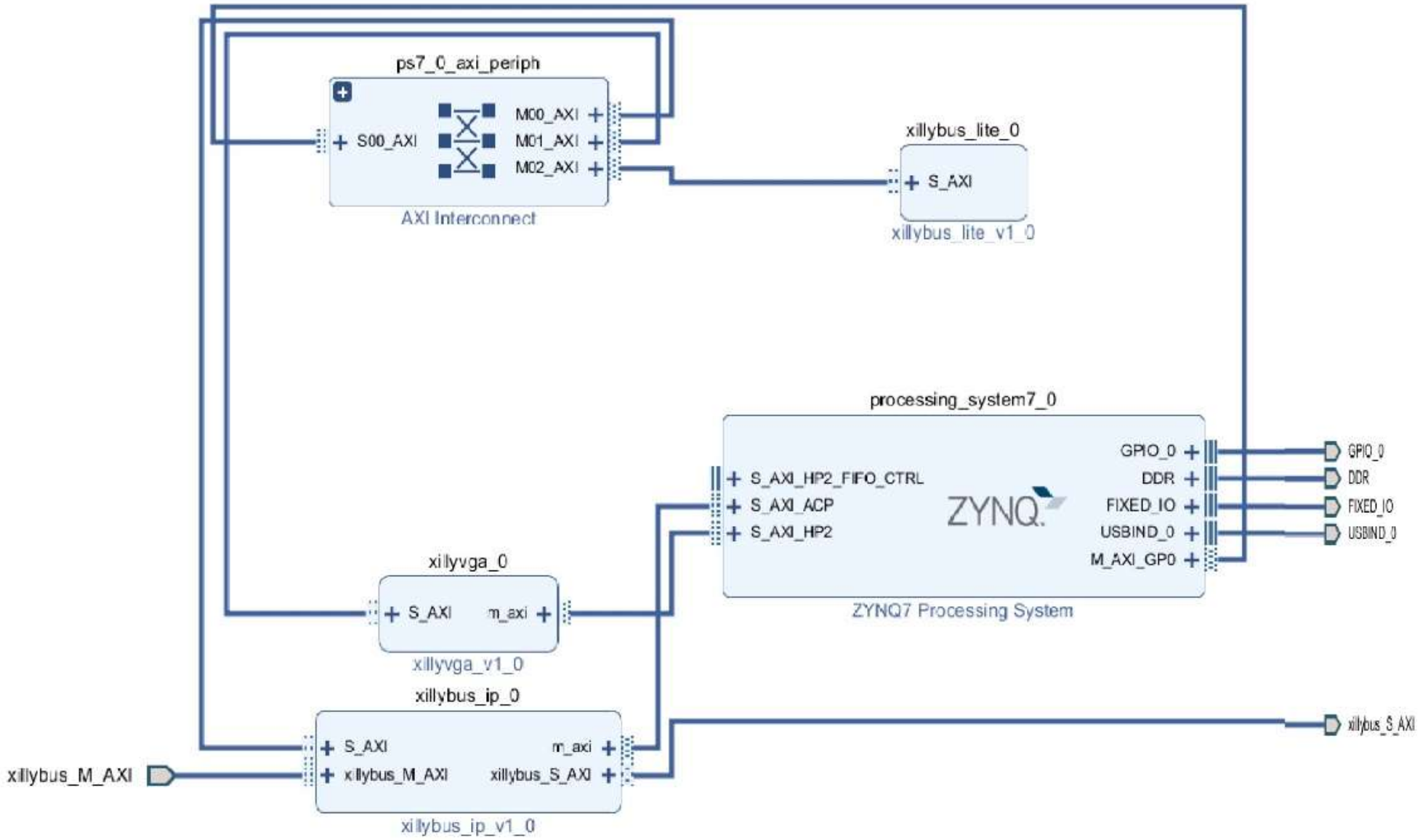
[5] http://zedboard.com/misc/files/ZedBoard_RevC.1_Gerbers_preliminary.pdf



	FECHA	NOMBRE
Dibujado	12/06/2019	Alberto Otaño Jimenez
Comprobado		Alberto Otaño Jimenez
Dib. S. Norma	U.N.E	Tolerancia general

ESCUELA TECNICA SUPERIOR DE INGENIERIA INDUSTRIAL
Universidad de La Rioja
GRADO EN INGENIERIA ELECTRONICA

Escalas S/E	Circuito de conversión de tensión de 0 a 5V a -0,5 a 0,5V	**NÚMERO: 1**
	Entrada al conversor XADC	Sustituye a:
		Sustituido por:



	FECHA	NOMBRE
Dibujado	12/06/2019	Alberto Otaño Jimenez
Comprobado		Alberto Otaño Jimenez
Dib. S. Norma	U.N.E.	Tolerancia general

ESCUELA TECNICA SUPERIOR DE INGENIERIA INDUSTRIAL
Universidad de La Rioja
GRADO EN INGENIERIA ELECTRONICA

Escalas S/E

Diagrama de bloques de la parte hardware

Esquema de las conexiones del XADC con la FIFO con Xillybus

NUMERO: 2

Sustituye a:

Sustituido por:

UNIVERSIDAD DE LA RIOJA

Trabajo Fin de Grado

Desarrollo de un “Datalogger” basado en ARM Cortex A9 y Linux mediante dispositivo Zynq-7000

5. PLIEGO DE CONDICIONES

Alberto Otaño Jiménez

Julio 2019

PLIEGO DE CONDICIONES

1. INTRODUCCIÓN

El objeto de este Pliego de Condiciones es recoger y fijar las disposiciones técnicas, administrativas y económicas, y las normativas que ha de regir la instalación, suministro y mantenimiento del presente proyecto.

Las características del diseño de este proyecto han sido descritas en detalle en la memoria de este proyecto y en sus correspondientes anexos.

Las condiciones que se especifican en este documento tratan de cumplir con la calidad esperada para este proyecto. En caso de no realizarse según estas condiciones, el proyectista no se responsabilizará de los fallos o averías que puedan ocasionarse en su funcionamiento, y los problemas derivados repercutirían sobre terceras personas. Cualquier modificación sobre lo establecido en los documentos de este sistema deberán ser aprobados por el proyectista.

2. CONDICIONES GENERALES

El autor de este proyecto ha cursado los estudios de Grado en Ingeniería Electrónica Industrial y Automática en la Escuela Técnica Superior de Ingeniería Industrial de la Universidad de La Rioja. El autor conoce y cumple la normativa establecida sobre "Trabajo Fin de Grado" establecida por la universidad.

Este proyecto se trata de un prototipo, por lo que las especificaciones expuestas serán consideradas en el caso de implantar físicamente el prototipo en un caso real.

La propiedad intelectual del autor y tutor del Trabajo Fin de Grado se regirán según el artículo 12 del reglamento de Proyectos Fin de Carrera de la Escuela Técnica Superior de Ingeniería Industrial de La Rioja, aprobado por el Consejo de Gobierno el 15 de abril de 2005 que dicta: "Podrá ser consultada la copia de la biblioteca si el autor y el Director de Proyecto Fin de Carrera así lo autorizan por escrito"

Este proyecto se ajusta a los reglamentos y normativas electrónicas vigentes. Una vez terminado el proyecto, se podrán llevar a cabo las modificaciones pertinentes, pero siempre bajo la supervisión del proyectista.

3. CONDICIONES ADMINISTRATIVAS

El proyecto constará de los siguientes documentos:

- Un Índice General que indicará la página de comienzo de cada uno de los documentos que forman el proyecto.
- Una Memoria donde se considerarán las necesidades a satisfacer y los factores técnicos a tener en cuenta entrando en profundidad en las posibles soluciones técnicas y en la justificación de la solución elegida.
- Anexos donde se recogerá la documentación considerada de interés para ampliar la descripción detallada de los componentes del sistema.
- Una serie de Planos que describirán los circuitos eléctricos de cada uno de los componentes hardware que se integran en el sistema y los planos de sus correspondientes circuitos impresos, los cuales deberán servir para la perfecta realización de los componentes del sistema.
- Pliego de Condiciones, este documento en el que se establecen las diferentes condiciones técnicas, económicas y administrativas para que proyecto pueda materializarse, evitando posibles malinterpretaciones.
- Presupuesto donde se recogerá el coste de todos los componentes utilizados y la suma total que, junto a la mano de obra, dará el coste final del proyecto. Dicho presupuesto contiene la valoración económica global, desglosada y ordenada por partidas.

4. NORMATIVA Y REGLAMENTACIÓN

El proyecto está supeditado tanto a la normativa española como a normas de uso europeo e internacional.

4.1. Normativa relativa a productos electrónicos

Respecto al desarrollo de productos electrónicos, se puede en encontrar en AENOR (Asociación Española de Normalización y Certificación) las siguientes normativas:

- Norma UNE1302--2:1973. Vocabulario electrotécnico. Electrónica.
- Norma UNE-EN611000-4-3-1998. Compatibilidad electromagnética.
- UNE 20-050-74. Código para las marcas de resistencia y condensadores. Valores y tolerancias.
- UNE 20-531-73. Series de colores nominales para resistencias y condensadores.

La RoHS es una directiva de la UE que restringe el uso de 6 materiales peligrosos en la fabricación de diversos tipos de equipos eléctricos y electrónicos, obligando a los fabricantes a demostrar que sus productos contienen valores de concentración por debajo de los valores de concentración máximos (VCM), en las siguientes sustancias: plomo, mercurio, cadmio, cromo hexavalente, bifenilos policromados (PBB) y éter de bifenilo o policromado (PBDE).

Es una directiva de "mercado único", es decir, establece estándares de productos y se aplica a todos los estados miembros, debiéndose implantar de la misma manera en todos ellos.

4.2. Normativa relativa a materiales y equipos

Los materiales y equipos de este proyecto deben cumplir los estándares nacionales e internacionales en vigor y de obligado cumplimiento. Entre otros:

- UNE 20-324 Grados de protección de los envolventes del material eléctrico de baja tensión.
- UNE 20-334 Conductos para instalaciones eléctricas.
- UNE 21-401 Conductores eléctricos aislados.
- UNE 21-402. Conductores eléctricos aislados y desnudos.

Asimismo, el proyecto se regirá por el Reglamento Electrotécnico de Baja Tensión, en el que se tendrán en cuenta las siguientes normativas:

- M.I. B.T.029 referida a instalaciones de baja tensión de menos de 50 voltios
- M.I. B.T.029 referida a condiciones generales de instalación, de utilización, y requisitos a cumplir en el diseño de instalaciones de baja tensión.

Además, deberá tenerse en cuenta la normativa de cumplimiento no obligatorio que establecen algunos fabricantes de materiales y componentes.

4.3. Normativa relativa a lenguajes de programación

Real Decreto 450/2010, de 16 de abril, por el que se establece el título de Técnico Superior en Desarrollo de Aplicaciones Multiplataforma y se fijan sus enseñanzas mínimas.

4.4. Normativa sobre elaboración de proyectos

La elaboración del presente proyecto está basada en la norma española UNE 157001:2014 elaborada en junio de 2014 "Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico".

5. CONDICIONES DE MATERIALES Y EQUIPOS

5.1. ZedBoard

La entrada principal de la placa es de 12V. Un convertidor de CA / CC compatible tendrá un diámetro interno de 2,5 mm y un diámetro exterior de 5,5 mm por donde se realiza la entrada de alimentación

Existe una resistencia de limitación de corriente de 10mΩ y 1W en serie con la fuente de alimentación de entrada de 12V. El jumper J21 monta esta resistencia para medir el voltaje a través de esta resistencia para calcular la potencia de ZedBoard.

ZedBoard incluye un interruptor de ON/OFF con el nombre de SW8, para la entrada de 12V. Cuando SW8 está en la posición OFF, la placa no recibe alimentación.

Existen varios reguladores Maxim que distribuyen la tensión a las diferentes vías de la tarjeta (Figura 1)

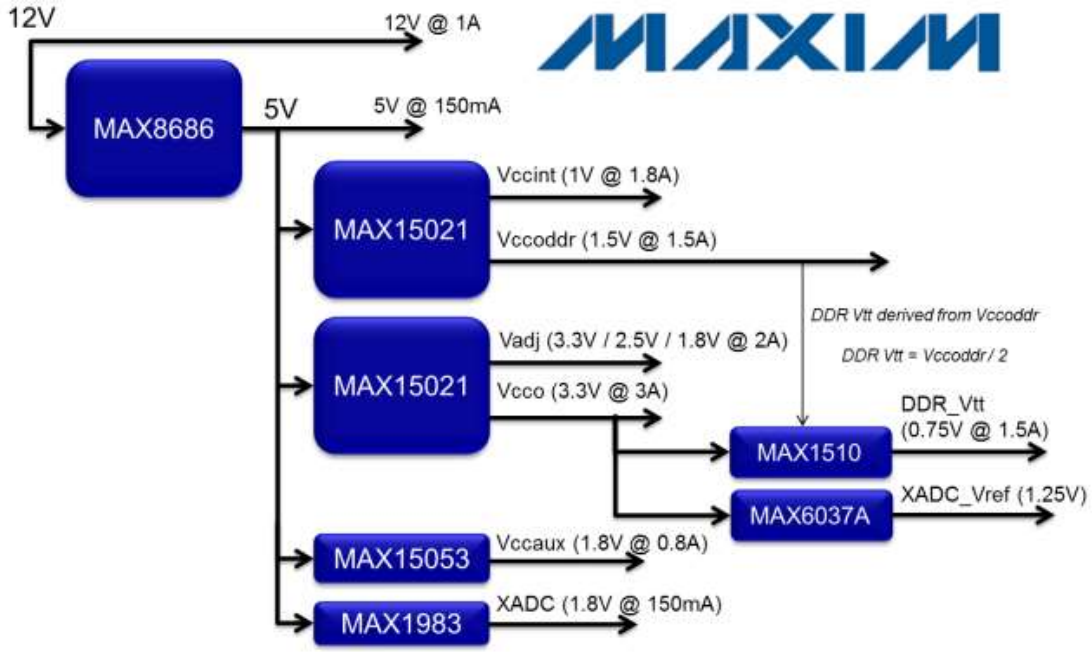


Figura 1: Distribución de la energía por reguladores Maxim

La siguiente Tabla 1 muestra los mínimos voltajes requeridos por cada entrada, la corriente y las tolerancias

Voltage (V)	Current (A)	Tolerance
1.0 (Vccint)	1.3	5.00%
1.5 (Vccoddr)	1.5	5.00%
1.8 (Vccaux)	0.8	5.00%
1.8, 2.5, 3.3 (jumper adjustable, 2.5V default) (Vadj)	2	5.00%
3.3 (Vcco/FMC/Pmod)	3	5.00%
1.8 (analog) (Vccadc)	0.15	5.00%
1.25 reference (Vrefp)	0.005	0.2%, 50ppm/°C
0.75 (DDR3 Vtt)	1.5	5.00%
5.0 (Filtered for XADC)	0.15	5.00%

Tabla 1:Tensión mínima, corriente y tolerancias de las diferentes entradas de la ZedBoard

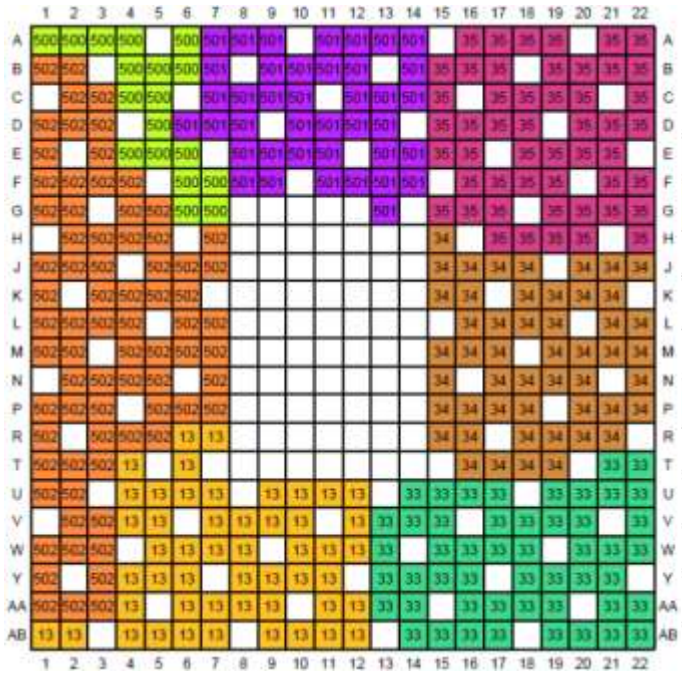
Un LED de color verde, LD13, indica cuando la alimentación es aceptada por la placa. La alimentación correcta está conectada con el indicador de reinicio y de programación evitar el funcionamiento de la placa cuando la alimentación no es buena.

La estimación de consumo por cada banco de la placa se muestra en la siguiente Tabla 2.

Feature	Part Number	MFG	Bank	1.0V	1.5V	1.8V	Vadj	3.3V
EPP PS	Z7020-CSG484	Xilinx		600	250	150		150
EPP PL	Z7020-CSG484	Xilinx		1200		300		50
DDR3	MT41K128M16HA-15E:D	Micron	VCCO_DDR		425			
DDR3 Termination					360			
QSPI Flash	S25FL256S Vcc	Spansion						100
QSPI Flash	S25FL256S Vio	Spansion	VCCO_MIO0					2
10/100/1000 PHY	88E1518 internal	Marvell		72		63		50
10/100/1000 PHY	88E1518 Vddo	Marvell	VCCO_MIO1			27		
USB 2.0 OTG PHY	TUSB1210	TI	VCCO_MIO1			30		30
HDMI Transmitter	ADV7511	ADI				181		0
Audio Codec	ADAU1761	ADI						58
OLED	UG-2832HSWEG04	Univision						28
Clock	F4100	Fox						40
USB JTAG	SMT1	Digilent						110
UART to USB	CY7C64225	Cypress	VCCO_MIO1					26
SD Card			VCCO_MIO1					100
PS User LED (green)			VCCO_MIO0					5
PL User LEDs (green) x8								40
PS User PB x2			VCCO_MIO0					0
PL User PBs x5								0
PL User Slide Switches x8								0
Status LEDs (amber) x4?								20
DONE LED (blue)								5
PS Pmod	TE 5-534206-6	TE	VCCO_MIO0					3000
PL Pmod #1	TE 5-534206-6	TE						
PL Pmod #2	TE 5-534206-6	TE						
PL Pmod #3	TE 5-534206-6	TE						
PL Pmod #4	TE 5-534206-6	TE						
FMC-LPC							2000	
Total				**1872**	**1035**	**751**	**2000**	**3814**
				1.0	**1.5**	**1.8**	**2.5**	**3.3**

Tabla 2:Estimación de consumo de los diferentes bancos de la ZedBoard

Los diferentes bancos se deben conectar a las tensiones de la Figura 2.



EPP PS-Side	
Bank	**Voltage (default)**
MIO Bank 0/500	3.3V
MIO Bank 1/501	1.8V
DDR	1.5V
EPP PL-Side	
Bank0	3.3V
Bank 13	3.3V
Bank 33	3.3V
Bank 34	Vadj (2.5V)
Bank 35	Vadj (2.5V)

Figura 2: Izquierda: asignación del banco según el pin de la Zynq. Derecha: tensión a conectar cada banco.

La FPGA con la que se cuenta para este trabajo, pertenece a la familia Zynq-7000, más concretamente, se trata del dispositivo XC7Z020-CLG484-1. Se trata de la parte más sensible de toda la tarjeta ZedBoard, por lo que la temperatura se ve limitada a los rangos que ofrece dicha FPGA. El rango de temperatura en los cuales el dispositivo funciona correctamente es de 0ºC a 85ºC, como se observa en la Figura 3.

Device	Speed Grade and Junction Temperature Range		
	Commercial (C) 0°C to +85°C	Extended (E) 0°C to +100°C	Industrial (I) –40°C to +100°C
XC7Z007S XC7Z012S XC7Z014S	-1	-2	-1, -2
XC7Z010 XC7Z015 XC7Z020	-1	-2, -3	-1, -2, -1L
XC7Z030 XC7Z035 XC7Z045	-1	-2, -3	-1, -2, -2L
XC7Z100	-1	-2	-1, -2, -2L

Figura 3:Rango de temperaturas de los diferentes dispositivos Zynq-7000.

Dado que el prototipo se trata de un *Datalogger* que se dejara desatendido en algún lugar remoto, soportando condiciones que pueden ser adversas, se recomienda su colocación dentro de una Caja de protección de 230x180x85mm como la que se adjunta a continuación en la Figura 4, y que se puede adquirir en el siguiente enlace: https://www.efectoled.com/es/comprar-cajas-de-registro-y-empalme/4509-caja-estanca-

230x180x85-mm.html?gclid=Cj0KEQjw8-LnBRCyxtfMl-Cbu48BEiQA6eUMGl_ctlaRhh5HYLCGXTY0c5XKkIGyKBHqM6Bpon0uzfEaAm8j8P8HAQ.

Figura 4:Caja de protección para ZedBoard.

5.2. Conversor analógico-digital XADC

Los rangos de tensión de entrada que soportan los canales del conversor ADC varían desde:

- -0,5 V a 0,5 V en el caso de haber sido configurados en modo bipolar
- 0 V a 1 V en el caso de haber sido configurados en modo unipolar.

Para la realización del prototipo y las pruebas, es necesario contar con una fuente de alimentación capaz de proporcionar dichas tensiones.

El conversor solo tiene tres entradas físicas, las cuales pueden recibir tensión analógica de entrada. Dichas entradas son: VpVn, Vaux0 y Vaux8. Dado que el XADC ofrece la opción de tener hasta diecisiete entradas de conversión (una dedicada y dieciséis auxiliares), se debe utilizar el modo multiplexor externo para esta opción. Este modo multiplexa todas las entradas seleccionadas, y la salida de estos valores es el canal dedicado VpVn. Los canales seleccionados se irán leyendo en orden y la salida de los valores convertidos se irá produciendo de forma progresiva por el canal dedicado VpVn.

La máxima velocidad que puede muestrear el conversor es de 26 MHz y la mínima es de 1 MHz. El conversor tiene dos relojes: DCLK para el DRP y ADCCLK para el funcionamiento interno. El primero tiene una frecuencia de entre 8 y 250 MHz (utilizados para la comunicación DRP) y el segundo oscila entre 1 y 26 MHz, tratándose de la frecuencia de muestreo.

Se ha añadido más información de interés sobre los modos de funcionamiento, pin-out, registros, etc del XADC en el apartado de la Memoria 8.1.3. XADC

5.3. Conexiones

Se requieren varias conexiones de la tarjeta ZedBoard hacia componentes externos para la puesta en marcha del sistema. Las conexiones que hay que realizar son las siguientes:

- **VGA**: Es necesario contar con un conector VGA conectado a la salida VGA de la ZedBoard y a la entrada de un monitor LCD.
- **Ethernet**: Se requiere conexión a internet para instalar ciertas librerías y dependencias en la tarjeta. Es necesario contar con un router que disponga de conexión a internet.
- **Alimentación**: En el kit de compra de la placa ZedBoard existe un transformador para convertir la tensión de red en los 12V de entrada de la placa (5.1. ZedBoard)
- **Tarjeta SD**: Donde se alojará el sistema operativo Xillinux. En la placa ZedBoard existe una ranura para tarjetas SD.
- **Conexión USB**: Dado que la tarjeta ZedBoard cuenta con una salida USB OTG, es necesario disponer de un cable adaptador micro USB-OTG a USB (Figura 5)

Figura 5:Cable micro USB-OTG a USB

Además, para realizar las conexiones a los diferentes elementos de entrada (ratón, teclado, pendrive, etc) se requiere un ladrón multipuerto USB. A través de él son necesarias las siguientes conexiones:

  - Ratón
  - Teclado
  - Pendrive

- **Cables de cobre**: Para realizar las pruebas de entrada al conversor XADC, ha sido necesario disponer de jumpers con configuración macho-hembra y hembra-hembra.

5.4. Software

Para la realización de este proyecto es necesario disponer de un ordenador capaz de soportar los sistemas operativos:

- Windows 10

- Linux Ubuntu 14.04 LTS
- Xillinux (en la tarjeta SD de la ZedBoard)

Dentro de Windows 10 se requieren programas como:

- Vivado 2018.2 WebPack Suite
- USB Image Tool
- Linux Live USB Creator
- Microsoft Excel 2016

En Linux Ubuntu ha sido necesaria la instalación de programas como:

- Qt Creator
- Qt Designer
- Spyder
- Libre Office Calc
- Librerías como:
  - Tkinter
  - Setuptools
  - Suds
  - Matplotlib
  - Numpy
  - Scipy
  - Imagetk
  - Patsy
  - Pandas
  - Fastcluster
  - Statistics
  - email

Dentro de Xillinux ha sido también necesaria la instalación de todas las librerías de Ubuntu. Para la instalación de todo este software, se ha requerido conexión a internet, bien mediante conexión Wifi en el caso de Ubuntu y Windows o mediante cable Ethernet en el caso de la FPGA.

6. CONDICIONES DE EJECUCIÓN Y MONTAJE

6.1. Condiciones de configuración

La aplicación permite la programación temporal de la adquisición de datos de tres sensores: temperatura, humedad y nivel para un pluviómetro. Esta configuración se puede realizar de dos modos:

Figura 6: Menú configuración de la aplicación

- Programación por tiempo: Se podrá elegir los días de la semana que quiere que se realice la captura de datos y la hora. Por ejemplo, se puede tomar la temperatura todos los lunes, miércoles y viernes a las 8:30.
- Programación por periodo: Se puede ajustar un tiempo de espera entre captura y captura. Por ejemplo, se puede muestrear la humedad relativa cada 30 minutos y guardar ese valor.

También es necesario tener en cuenta que el formato de salida es CSV, ya que puede ser interpretado por cualquier hoja de cálculo. Se generarán tres archivos de salida en la carpeta “CSV” dentro del directorio que contiene el programa principal. Estos ficheros de salida pueden ser sobrescritos cada vez que se inicie la adquisición y registro de datos o modificados. Al sobrescribirlos, se borrarán los datos ya existentes en dicho fichero, sin embargo, al modificarlo, los datos se añadirán a dichos ficheros. Para ajustar dicha configuración existe una opción en la pantalla del Menú Configuración de la aplicación (Figura 6)

6.2. Puesta en marcha y mantenimiento preventivo

Para el correcto funcionamiento de este sistema, necesita los elementos externos que se enumeran en el apartado 5.3. Conexiones:

- Pantalla LDC con cable VGA

- Transformador de tensión de red a 12 V en la entrada de alimentación de la ZedBoard
- Cable micro USB-OTG a USB, conectado a un ladrón de USB, conectado a su vez a:
    - Teclado
    - Ratón
    - Pendrive
- Conexión Ethernet con un router

Tras realizar todas las conexiones, se deberá poner en ON el Switch SW8, para que proceda a alimentar a la tarjeta ZedBoard. El siguiente paso es activar el modo grafico de Xillinux mediante el comando:

```
startx
```

Al ladrón de USB, se debe conectar un Pendrive o un disco duro externo donde se quiere que se almacenen la información captada por el *Datalogger*. A mayor capacidad del dispositivo USB conectado, mayor cantidad de información podrá almacenar.

Luego se ejecutará el fichero DATALOGGER2.py y se abrirá la aplicación de interfaz de usuario para que se proceda a la configuración de la programación temporal y el inicio del registro de datos.

El circuito de prueba, con el que se simularan los sensores mencionados anteriormente, consta de los siguientes elementos para cada canal, por lo tanto, hay que multiplicar todos los elementos enumerados por tres canales:

- 1 x Amplificador Operacional TL082
- 2 x Resistencias de 1K
- 1 x Resistencia de 10K
- 1 x Resistencia de 4,7K
- 1 x Potenciómetro (simulando al sensor correspondiente)

Dado que el operacional TL082 cuenta con dos entradas inversoras, dos no inversoras y dos salidas, se puede utilizar uno solo para dos canales.

El circuito estará alimentado a una tensión de 5V, que transformará a +0,5V de máxima y -0,5V de mínima. Las salidas de este circuito irán a los pines correspondientes del XADC. El circuito a realizar para cada sensor es el que se muestra en la Figura 7.

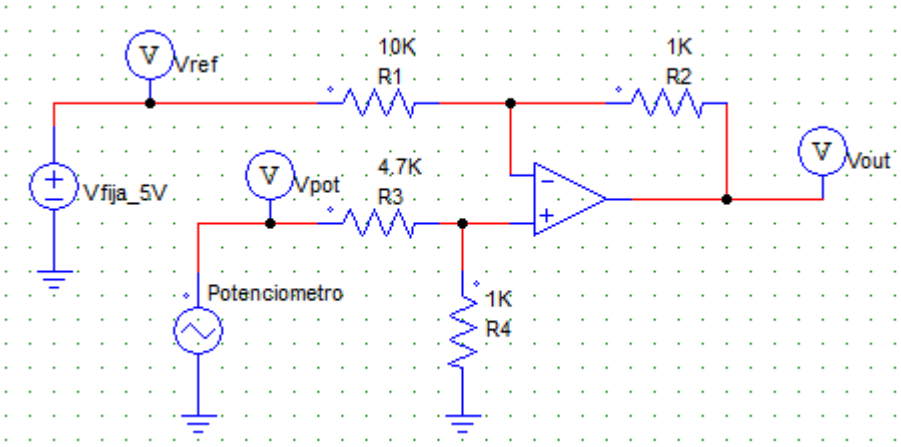


Figura 7:Circuito de simulación de cada canal.

Es recomendable realizar periódicamente una revisión de los archivos de salida CSV para comprobar que son datos correctos. Se proporcionarán dos años de garantía cubriendo los fallos derivados del funcionamiento normal de este, no cubriendo daños producidos por una mala utilización.

En caso de utilizar un dispositivo USB externo para el almacenamiento de los datos, es necesario formatear dicho dispositivo con el nombre “ALBERTO” y crear un directorio para el almacenamiento de datos dentro del USB con la ruta /DATALOGGER2_tarjeta/CSV/

7. CONDICIONES ECONÓMICAS

7.1. Errores en el proyecto

Ante cualquier fallo o error tanto de diseño como de funcionamiento se informará al proyectista para poder dar solución al problema.

En este caso, se recomienda la desconexión inmediata del sistema para evitar posibles daños, tanto de los propios componentes como de componentes externos, hasta que el problema se vea resuelto.

7.2. Liquidación

Terminada la instalación del sistema, se procederá a la liquidación final, en la que se incluye el importe de las unidades de realización, así como las posibles modificaciones del proyecto que hayan sido aprobadas por la dirección del proyecto.

Al suscribir el contrato, el cliente habrá de abonar el 80% del presupuesto. El 20% quedara como garantía durante los seis primeros meses, a partir de la fecha de puesta en marcha del sistema.

Si transcurridos seis meses desde la puesta en marcha no se ha manifestado ningún defecto o error de funcionamiento, el cliente abonara el 20% que estaba

pendiente. A partir de ese momento, se considerarán concluidos los compromisos entre ambas partes, a excepción del periodo de garantía.

8. DISPOSICIÓN FINAL

Las partes contratantes, ingeniero Director y empresa cliente, se ratifican en el contenido del presente pliego de condiciones, que tiene igual validez, a todos los efectos, que una escritura pública, prometiendo su fiel cumplimiento.

UNIVERSIDAD DE LA RIOJA

Trabajo Fin de Grado

Desarrollo de un “Datalogger” basado en ARM Cortex A9 y Linux mediante dispositivo Zynq-7000

6. MEDICIONES

Alberto Otaño Jiménez

Julio 2019

MEDICIONES

Partida de proyecto CD01: Componentes electrónicos		
Código UP	**Descripción**	**Cantidad**
CD01.1	Placa ZedBoard con Zynq-7000 incorporada	1 u
CD01.2	Alimentador de 12 V	1 u
CD01.3	Tarjeta microSD de 32 GB + adaptador	1 u
CD01.4	Cable micro-USB	1 u
CD01.5	Adaptador USB OTG a USB estándar	1 u
CD01.6	Cable Ethernet	1 u
CD01.7	Conectores Jumpers macho-hembra	40 u
CD01.8	Conectores Jumpers hembra-hembra	40 u
CD01.9	Amplificador operacional TL082	2 u
CD01.10	Resistencia 1K	6 u
CD01.11	Resistencia 10K	3 u
CD01.12	Resistencia 4,7K	3 u
CD01.13	Potenciómetro	3 u
CD01.14	Protoboard	1 u
CD01.15	Fuente de alimentación Power Supply FAC-363	1 u
CD01.16	Cables con conexión banana-cocodrilo 3 hilos	1 u
CD01.17	Multímetro Digital Portátil	1 u

Partida de proyecto CD02: Equipos informáticos		
Código UP	**Descripción de la unidad de proyecto**	**Cantidad**
CD02.1	Ordenador portátil ASUS Intel Core i7	1 u
CD02.2	Ratón	2 u
CD02.3	Teclado	1 u
CD02.4	Pendrive USB 8GB	1 u
CD02.5	Ladrón Hub USB 2.0 4 puertos Trust	1 u

CD02.6	Pantalla LCD Airis	1 u
CD02.7	Cable VGA	1 u

Partida de proyecto CD03: Recursos humanos		
Código UP	**Descripción de la unidad de proyecto**	**Cantidad**
CD03.1	Horas de investigación	40 u
CD03.2	Horas de desarrollo hardware	100 u
CD03.3	Horas de desarrollo software	100 u
CD03.4	Horas de prueba y puesta en marcha	40 u
CD03.5	Horas de redacción	70 u

Partida de proyecto CD04: Licencias software		
Código UP	**Descripción de la unidad de proyecto**	**Cantidad**
CD04.1	Windows 10 Home – 64 bits	1 u
CD04.2	Microsoft Office 2016	1 u
CD04.3	Xilinx Vivado 2018.2	1 u
CD04.4	Ubuntu 14.04	1 u
CD04.5	Linux Live USB Creator	1 u
CD04.6	USB Image Tool	1 u

UNIVERSIDAD DE LA RIOJA

Trabajo Fin de Grado

Desarrollo de un “Datalogger” basado en ARM Cortex A9 y Linux mediante dispositivo Zynq-7000

7. PRESUPUESTOS

Alberto Otaño Jiménez

Julio 2019

PRESUPUESTOS

1. CUADRO DE PRECIOS UNITARIOS

Partida de proyecto CD01: Componentes electrónicos		
Código	**Descripción**	**Precio unitario**
CD01.1	Placa ZedBoard con Zynq-7000 incorporada	510,74 €
CD01.2	Alimentador de 12 V	9,84 €
CD01.3	Tarjeta microSD de 32 GB + adaptador	8,99 €
CD01.4	Cable micro-USB	2,21 €
CD01.5	Adaptador USB OTG a USB estándar	1,04 €
CD01.6	Cable Ethernet	6,27 €
CD01.7	Conectores Jumpers macho-hembra	4,99 €
CD01.8	Conectores Jumpers hembra-hembra	4,99 €
CD01.9	Amplificador operacional TL082	0,56 €
CD01.10	Resistencia 1K	1,24 €
CD01.11	Resistencia 10K	1,24 €
CD01.12	Resistencia 4,7K	1,24 €
CD01.13	Potenciómetro	1,94 €
CD01.14	Protoboard	3,20 €
CD01.15	Fuente de alimentación Power Supply FAC-363	400 €
CD01.16	Cables con conexión banana-cocodrilo 3 hilos	2,42 €
CD01.17	Multímetro Digital Portátil	8,95 €

Partida de proyecto CD02: Equipos informáticos		
Código	**Descripción**	**Precio unitario**
CD02.1	Ordenador portátil ASUS Intel Core i7	850 €
CD02.2	Ratón Trust	9,95
CD02.3	Teclado Logitech K120	17,99 €
CD02.4	Pendrive USB 8GB	5,34 €

CD02.5	Ladrón Hub USB 2.0 4 puertos Trust	9,99 €
CD02.6	Pantalla monitor LCD Airis	60 €
CD02.7	Cable VGA	2,40 €

Partida de proyecto CD03: Recursos humanos		
Código	**Descripción**	**Precio unitario**
CD03.1	Horas de investigación	20 €
CD03.2	Horas de desarrollo hardware	24 €
CD03.3	Horas de desarrollo software	24 €
CD03.4	Horas de prueba y puesta en marcha	20 €
CD03.5	Horas de redacción	16 €

Partida de proyecto CD04: Recursos humanos		
Código	**Descripción**	**Precio unitario**
CD04.1	Windows 10 Home – 64 bits	22,99 €
CD04.2	Microsoft Office 2016	69 €
CD04.3	Xilinx Vivado 2018.2	0 €
CD04.4	Ubuntu 14.04	0 €
CD04.5	Linux Live USB Creator	0 €
CD04.6	USB Image Tool	0 €

2. PRESUPUESTOS PARCIALES

Presupuesto parcial CD01: Componentes electrónicos				
Código	**Descripción**	**Precio unitario**	**Cantidad**	**Precio parcial**
CD01.1	Placa ZedBoard con Zynq-7000 incorporada	510,74 €	1	510,74 €
CD01.2	Alimentador de 12 V	9,84 €	1	9,84 €
CD01.3	Tarjeta microSD de 32 GB	8,99 €	1	8,99 €

CD01.4	Cable micro-USB	2,21 €	1	2,21 €
CD01.5	Adaptador USB OTG a USB estándar	1,04 €	1	1,04 €
CD01.6	Cable Ethernet	6,27 €	1	6,27 €
CD01.7	Conectores Jumpers macho-hembra	4,99 €	1	4,99 €
CD01.8	Conectores Jumpers hembra-hembra	4,99 €	1	4,99 €
CD01.9	Amplificador operacional TL082	0,56 €	2	1,12 €
CD01.10	Resistencia 1K	1,24 €/100 u	1	1,24 €
CD01.11	Resistencia 10K	1,24 €/100 u	1	1,24 €
CD01.12	Resistencia 4,7K	1,24 €/100 u	1	1,24 €
CD01.13	Potenciómetro	1,94 €	1	1,94 €
CD01.14	Protoboard	3,20 €	1	3,20 €
CD01.15	Fuente de tensión Power Supply FAC 363	400 €	1	400 €
CD01.16	Cables con conexión banana-cocodrilo 3 hilos	2,42 €	1	2,42 €
CD01.17	Multímetro Digital Portátil	8,95 €	1	8,95 €
Total CD01				**970,42 €**

Asciende la citada partida CD01 de Componentes Electrónicos a la cantidad de NOVECIENTOS SETENTA EUROS CON CUARENTA Y DOS CÉNTIMOS.

Unidad de proyecto CD02: Equipos informáticos				
Código	**Descripción**	**Precio unitario**	**Cantidad**	**Precio parcial**
CD02.1	Ordenador portátil ASUS Intel Core i7	850 €	1	850 €
CD02.2	Ratón Trust	9,95	2	19,90
CD02.3	Teclado Logitech K120	17,99 €	1	17,99 €
CD02.4	Pendrive USB 8GB	5,34 €	2	10,68 €

CD02.5	Ladrón Hub USB 2.0 4 puertos Trust	9,99 €	1	9,99 €
CD02.6	Pantalla monitor LCD Airis	60 €	1	60 €
CD02.7	Cable VGA	2,40 €	1	2,40 €
Total CD02				**970,96 €**

Asciende la citada partida CD02 de Equipos Informáticos a la cantidad de NOVECIENTOS SETENTA EUROS CON NOVENTA Y SEIS CÉNTIMOS.

Unidad de proyecto CD03: Recursos humanos				
Código	**Descripción**	**Precio unitario**	**Cantidad**	**Precio Parcial**
CD03.1	Horas de investigación	20 €	40	800 €
CD03.2	Horas de desarrollo hardware	24 €	100	2400 €
CD03.3	Horas de desarrollo software	24 €	100	2400 €
CD03.4	Horas de prueba y puesta en marcha	20 €	40	800 €
CD03.5	Horas de redacción	16 €	70	1120 €
Total CD03				**7.520,00 €**

Asciende la citada partida CD03 de Recursos Humanos a la cantidad de SIETE MIL QUINIENTOS VEINTE EUROS.

Unidad de proyecto CD04: Licencias software				
Código	**Descripción**	**Precio unitario**	**Cantidad**	**Precio Parcial**
CD04.1	Windows 10 Home – 64 bits	22,99 €	1	22,99 €
CD04.2	Microsoft Office 2016	69 €	1	69 €
CD04.3	Xilinx Vivado 2018.2	0 €	1	0 €
CD04.4	Ubuntu 14.04	0 €	1	0 €

CD04.5	Linux Live USB Creator	0 €	1	0 €
CD04.6	USB Image Tool	0 €	1	0 €
Total CD04				**91,99 €**

Asciende la citada partida CD04 de Licencias Software a la cantidad de NOVENTA Y UN EUROS CON NOVENTA Y NUEVE CÉNTIMOS.

3. PRESUPUESTO DE EJECUCIÓN MATERIAL

Resumen general del presupuesto		
Código	**Nombre**	**Precio Parcial**
CD01	Componentes electrónicos	970,42 €
CD02	Equipos informáticos	970,96 €
CD03	Recursos humanos	7520,00 €
CD04	Licencias software	91,99 €
Presupuesto Ejecución Material		**9.553,37 €**

Asciende el citado Presupuesto de Ejecución Material a la cantidad de NUEVE MIL QUINIENTOS CINCUENTA Y TRES EUROS CON TREINTA Y SIETE CÉNTIMOS.

4. PRESUPUESTO DE EJECUCION POR CONTRATA

Presupuesto de ejecución material	9.553,37 €
13 % de gastos generales	1.241,94 €
6 % de beneficio industrial	573,20 €
Suma	11.368,51 €
21 % de IVA	2.387,39 €
Presupuesto de ejecución por contrata	13.755,90 €

Asciende el presupuesto de ejecución por contrata a la expresada cantidad de TRECE MIL SETECIENTOS CINCUENTA Y CINCO EUROS CON OCHENTA Y NUEVE CÉNTIMOS

Notas al presupuesto: Precios vigentes a junio de 2019, para una unidad de *Datalogger*, no contempla descuentos por volumen de compra. Si transcurre más de 12 meses, deberán revisarse.

FIRMADO:

D. Alberto Otaño Jiménez

Logroño a 27 de junio de 2019